Grado en Ingeniería Informática

Trabajo Final de Grado

# Generador automático de administradores de bases de datos

*Autor:*
Irene Roures Villalonga

*Supervisor:*
Sergio Aguado González
*Tutor académico:*
María del Mar Marcos López

Fecha de lectura: 18 de julio de 2017
Curso académico 2016/2017

## Resumen

En este Trabajo Final de Grado se ha desarrollado un sistema para generar de forma automática administradores web de bases de datos con funcionalidades para operar con los datos o instancias (filas) de las distintas entidades (tablas) de la base de datos. Concretamente los administradores disponen de funcionalidades para crear, modificar, ver los detalles (atributos), buscar, listar y eliminar instancias, para cada una de las entidades de la base de datos

Al quedarse obsoleto el sistema que se usaba previamente, la empresa decidió crear un sistema nuevo y actualizado que mejora el anterior, añadiendo nuevas características y funcionalidades que faltaban. El generador se ha programado con PHP y Twig para la generación y Angular2 para la implementación de las funcionalidades específicas del administrador.

Esta aplicación web es usada a la hora de crear la base de cualquier proyecto. El equipo de desarrollo de la empresa Cuatroochenta generará el administrador una vez se haya determinado la estructura de los datos de la aplicación. Este administrador generado por defecto se puede modificar manualmente para que sea exactamente el que el cliente quiere. De esta forma todos los proyectos parten con un entorno probado que satisface las necesidades del cliente.

## Palabras clave

Generación de código, administradores de bases de datos, Angular2, aplicaciones web.

## Keywords

Code generation, database administrators, Angular2, web applications.

# Índice general

# Índice de figuras

# Índice de cuadros

# Capítulo 1

# Introducción

## 1.1. Contexto y motivación del proyecto

El proyecto que se presenta en este documento se ha desarrollado para la empresa Cuatroochenta. Se trata de una empresa de creación de software a medida dirigido a otras compañías, cuya sede principal está establecida en el Espaitec de la Universitat Jaume I (UJI) en Castellón. Esta empresa, con cinco años de antigüedad, consta de una plantilla joven de más de 60 trabajadores formada por estudiantes y trabajadores con cierta experiencia. La empresa se caracteriza por fomentar la participación de su plantilla en toda clase de eventos dedicados al software y al emprendimiento. Cuatroochenta se dedica a desarrollar una gran variedad de tipos de software, tanto web como móvil, recientemente se decidió entrar también en el campo de los videojuegos. Este proyecto se encuentra dentro del ámbito de las aplicaciones web.

El proyecto consiste en desarrollar un sistema de creación automática de administradores. Partiendo de unos ficheros XML, donde se muestra el modelo de datos de la aplicación, se van a crear un conjunto de administradores cuya función será la gestión de una base de datos diferente para cada uno. Se pretende obtener un administrador con las funcionalidades que vienen definidas por el acrónimo CRUD (cuyas siglas en inglés responden a create, read, update y delete) generado automáticamente para cualquier proyecto y que se encargue de la persistencia de la información de su base de datos relacional. También se estableció desde el principio que debía ser lo más modular y desacoplado posible, para que si en un futuro se querían añadir módulos en otros lenguajes de programación (como por ejemplo nodeJS), se pudiera hacer sin perder mucho tiempo en ello y además que no dependiera de ninguna manera de la base de datos. La comunicación con la base de datos se hará mediante una API Rest.

Esta aplicación surge de la necesidad de ahorrar horas de programación repetitiva al comienzo de cualquier proyecto. En todo proyecto se debe crear un entorno de desarrollo y otro de prueba donde testear y probar que la aplicación funcione correctamente. La aplicación automatiza esta fase, por lo que los programadores partirán con los entornos ya creados y probados, únicamente tendrán que personalizarlos de la forma en que el cliente lo desee, ahorrando una gran cantidad de tiempo.

La empresa ya contaba con un sistema parecido al que se ha desarrollado, pero ineficiente y antiguo. Cada vez que se intentaba usar, salía un nuevo error que se tenía que solucionar lo más rápido posible para que siguiera funcionando. La gran cantidad de parches y el uso de lenguajes de programación desactualizados sin soporte por parte de la comunidad de programadores o empresas dedicadas a ello, hacían del programa poco mantenible y muy inestable con propensión a errores constantes. Debido a esto, se tenía que generar el administrador manualmente, sin usar la aplicación que más que ayudarles los retrasaba en su tarea.

## 1.2. Objetivos del proyecto

El objetivo principal del proyecto es desarrollar un software basado en Angular2 cuya función es la generación automática de administradores cada uno de los cuales se dedicará a la gestión de una base de datos diferente. Este sistema pretende ser una ayuda tanto para los trabajadores, que empezarán con una base probada y no desde cero, como para los clientes, quienes tendrán la posibilidad de ver el proyecto que ellos quieren implementar funcionando prácticamente desde el mismo momento en el que se establezca una estructura de datos. Con este proyecto ayudamos a incorporar al cliente dentro de la fase de desarrollo, opinando sobre el diseño y funcionalidad de lo que quiere exactamente, evitando errores a la hora de recoger requisitos y permitiendo que el cliente se haga una idea más clara del resultado que va a obtener al final del proceso.

Como objetivos específicos podemos destacar los que se describen a continuación:

- Diseñar una interfaz que permita la gestión de todos los datos de la base de datos. Al tratarse de una aplicación usada constantemente por todos los trabajadores de la entidad encargados de las aplicaciones web y clientes, se pretende crear una GUI amigable y fácil de usar con todos los elementos de los que se disponen.
- Analizar los requisitos que se quieren mejorar con respecto a la herramienta actual para que el nuevo sistema se adapte a las necesidades de la empresa.
- Diseñar un conjunto de etiquetas para una generación automática personalizada del administrador.

## 1.3. Estructura de la memoria

Una vez introducido el contexto y los objetivos del proyecto, el resto de la memoria está dividida en cinco capítulos.

El segundo capítulo es la “Descripción del proyecto”, donde se detallan todas las tecnologías usadas para el desarrollo del proyecto, es decir, lenguajes de programación y herramientas que han sido necesarias. También se explica la situación inicial del proyecto.

El tercer capítulo es la “Planificación del proyecto”, en él se explica cómo se ha organizado el proyecto paso a paso. Se comentan los métodos que se han seguido para la obtención de

requisitos a alto nivel y la metodología usada para la planificación y seguimiento del proyecto. También se detallan los recursos económicos usados durante su desarrollo.

El cuarto capítulo es el de "Análisis y diseño del sistema", en este capítulo se especifican y agrupan los requisitos según su tipo, se explica con detalle el fichero XML, que contiene los datos de entrada del programa, la estructura de los datos y la interfaz de usuario de un ejemplo concreto.

El quinto capítulo es la "Implementación y prueba", aquí se explica el entorno de desarrollo y lo necesario para la ejecución satisfactoria de la aplicación, la conexión con la base de datos y los módulos externos usados. También se detallan las pruebas de validación y verificación efectuadas.

El sexto capítulo es la "Conclusión", en él se presentan las conclusiones y futuras mejoras.

# Capítulo 2

# Descripción del proyecto

## 2.1. Tecnologías usadas

En este punto se van a explicar todas las herramientas usadas en el proceso de desarrollo del proyecto. El uso de Angular2 ha sido un requisito por parte de la empresa para la que se desarrolla el producto. Este framework ha sido usado para la funcionalidad de los administradores, junto con HTML, CSS y TypeScript. Para la generación automática de administradores se ha usado PHP y Twig.

Otras herramientas usadas han sido: WebStorm como programa para la generación de código, Git para el control de versiones y Bitbucket como sitio de almacenamiento del repositorio. Finalmente, se ha investigado el programa llamado Guiffy como software para unir ficheros con diferente contenido en uno solo.

### 2.1.1. Angular 2

Angular2 es un framework de JavaScript que permite crear aplicaciones web de una sola página, llamadas en inglés Single-Page-Applications (SPAs). Aunque si hay diferentes páginas con sus rutas correspondientes, se considera SPA ya que la página nunca se recarga. Siempre hay una parte de la aplicación que nunca cambia.

Además, permite que las diferentes páginas de las que consta la aplicación sean renderizadas en la parte del cliente. Esto aporta rapidez a la aplicación web ya que el servidor no la tiene que renderizar completamente cada vez que se cambian pequeñas partes de la aplicación, por lo que los cambios son mostrados instantáneamente.

En la figura 2.1 se muestra la estructura de cualquier aplicación Angular2. Angular2[1] se basa en componentes reutilizables mediante su identificador. Un componente es una combinación del modelo (o template) y del componente que le proporciona funcionalidad a la vista. Se le pueden añadir hojas de estilos que nos permiten una mayor personalización del modelo.

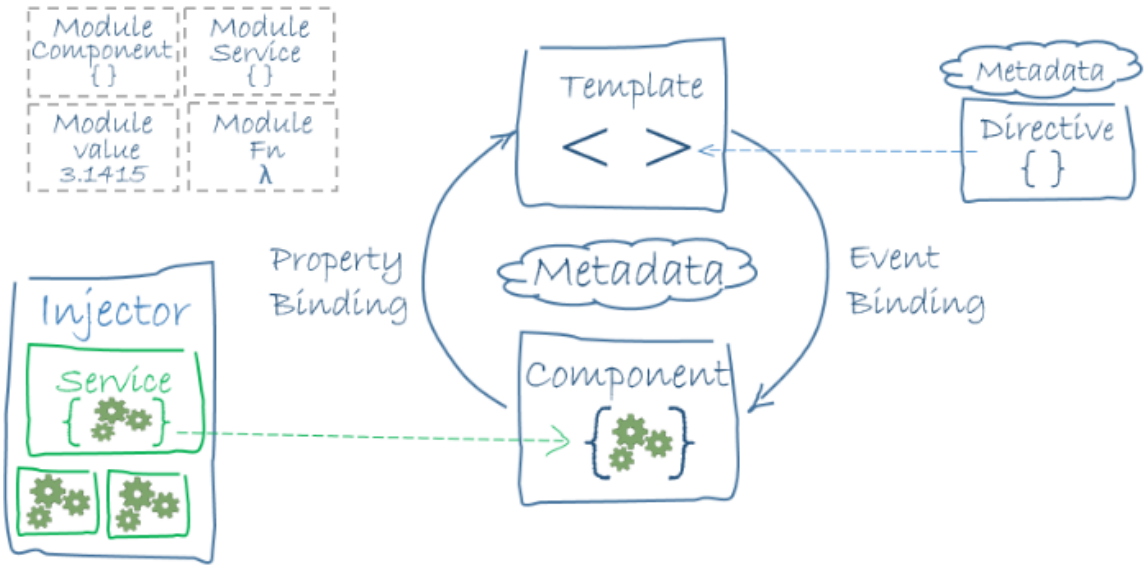


Figura 2.1: Estructura de las aplicaciones Angular2.
[1]

También se pueden inyectar servicios al componente. Los servicios permiten mantener el código desacoplado de módulos externos (como puede ser las peticiones a la base de datos) y evitar repetición de código (agrupan funcionalidades que se van a usar en muchas partes de la aplicación).

Una de las características de Angular2 es la posibilidad de data-binding en el modelo, lo que permite enlazar los elementos. Al declarar un elemento en el fichero TypeScript, se puede acceder a dicho elemento en el HTML. Si el elemento es un objeto compuesto de múltiples atributos, se pueden mostrar todos y cada uno de los atributos. También se puede modificar el elemento y atributos, de manera que lo que se cambia en la plantilla se cambia también en el fichero TypeScript. El data-binding permite cambios instantáneos en la plantilla y una mayor interacción entre el usuario y el modelo

Se le puede añadir cierta lógica al modelo mediante directivas. Con las directivas se pueden crear bucles, sentencias condicionales o variables de control. Las directivas hacen que los modelos sean dinámicos.

Además de componentes, plantillas y servicios, en la estructura de una aplicación Angular2 son indispensables los módulos y librerías. Como mínimo debe haber un módulo raíz por proyecto Angular2 denominado `app.module.ts`, en cuanto este fichero crezca se va a ir disgregando según funcionalidades. En este fichero se incluyen las declaraciones de todos los ficheros (ya sean componentes, modelos, servicios o módulos).

#### Angular CLI

En el proyecto se ha añadido Angular CLI. Se trata de una interfaz por línea de comandos que hace más cómoda y sencilla la gestión de aplicaciones Angular2. Permite crear un proyecto con la estructura de ficheros básica, lanzar la aplicación web, crear de forma automática ficheros en el proyecto y facilitar la tarea de añadir módulos o librerías externas.

### 2.1.2. TypeScript

TypeScript es un lenguaje de programación relativamente moderno, su antecesor es JavaScript de hecho TypeScript se compila internamente a JavaScript, por lo que cualquier código escrito en JavaScript es compatible con TypeScript (aunque no a la inversa). Sintácticamente son muy parecidos.

JavaScript es mucho menos intuitivo y cuesta mucho detectar los errores, pero si se tiene experiencia es más rápido crear cualquier programa y mucho más corto. TypeScript, por su parte, permite mantener el código organizado y es más intuitivo. Con TypeScript hay que declarar y poner el tipo a todas las variables y permite trabajar como un lenguaje orientado a objetos (con funciones, clases y atributos).

### 2.1.3. HTML5 y CSS3

HTML (HyperText Markup Language)[3] es el lenguaje de marcado usado en el modelo de los componentes de Angular2 que permite organizar el contenido de las páginas de la aplicación. CSS (Cascading Style Sheets) es un lenguaje para realizar las hojas de estilos que permiten personalizar todos los elementos de la interfaz y crear nuevos. Ambos lenguajes van muy relacionados en cuanto al diseño de la interfaz gráfica de usuario (GUI).

Para facilitar la tarea de hacer una interfaz gráfica de usuario clara y estéticamente agradable se ha hecho uso de una plantilla y del framework Bootstrap. Ambos elementos son una combinación de HTML junto con CSS.

Bootstrap provee la aplicación de elementos muy básicos, variados y estándares, sin embargo la plantilla está formada un conjunto de elementos pensados para ser usados en la misma página, con un diseño muy marcado y propio.

La plantilla se ha personalizado eliminando todos aquellos elementos que no se iban a usar, además, se han incorporado otros elementos que se habían creado previamente de forma manual y, se han modificado los elementos de la plantilla para que se adaptaran a lo que se quería.

La interfaz gráfica se ha hecho reactiva (responsive) para que se adapte a cualquier tamaño de pantalla y que se pueda ver en tablets y ordenadores con pantallas grandes sin que queden todos los elementos juntos y sobrepuestos en la página.

### 2.1.4. PHP

PHP (Hypertext Preprocessor) es un lenguaje enfocado al desarrollo de aplicaciones web especialmente recomendado para la creación de scripts y no tanto para las aplicaciones de escritorio[2]. En este proyecto se ha usado PHP por su extensión llamada SimpleXML.

Esta extensión incluye una serie de instrucciones que facilitan la lectura de ficheros XML y su conversión a objetos con los que poder trabajar de forma sencilla. Permite guardar la estructura de la información contenida en el fichero XML en un JSON, y a partir de él ir obteniendo los nodos de información relevante para la aplicación.

### 2.1.5. Twig

Twig [4] es un motor para la creación de plantillas. Está diseñado para ser usado junto con PHP, aunque en este proyecto se ha usado con Angular2 para la generación de toda la estructura de la aplicación.

Permite generar cualquier tipo de fichero, por convención reciben el nombre que se desee junto con la extensión del fichero original (.html o .ts en este caso), seguida por la extensión de Twig (.twig).

Trata los ficheros como si fueran texto plano, a estos ficheros se le pueden añadir ciertas instrucciones para generar las plantillas que al ejecutarlas dan lugar a ficheros en el lenguaje de programación deseado.

Incluye tres tipos de instrucciones, el data-binding (que permite mostrar el contenido de las variables), las instrucciones lógicas (como bucles o sentencias condicionales) y los comentarios (usados para documentar las plantillas). Existe la posibilidad de añadir filtros para que una variable se muestre con cierto formato (en minúsculas o para las fechas).

Es muy flexible y adaptado a todo tipo de situaciones. Al usar Twig con Angular2, se produce un conflicto en cuanto a que ambos frameworks soportan el data-binding. Para solucionarlo, Twig nos permite adaptar la sintaxis según las preferencias del programador siempre y cuando sea indicado previamente. No es muy recomendable esta práctica debido a que hace el código confuso y difícil de entender para una persona ajena al proyecto.

En las plantillas se pueden definir bloques de texto para poder reutilizarlos en otras plantillas y evitar la repetición de código, además de mantenerlo limpio y organizado. Twig permite la herencia, es decir una plantilla puede extender otra. Cuando una plantilla hereda de otra, la plantilla hija puede implementar o extender los bloques de código declarados en la del padre o puede dejarlos vacíos para que los implemente otra plantilla hija.

Lo normal en las aplicaciones que usan plantillas Twig es tener dos o tres niveles de herencia. En este proyecto se ha trabajado con herencia a dos niveles. El primer nivel es una plantilla general única para la aplicación (todas las otras plantillas heredan de esta), en ella se incluyen los elementos que definen cómo están organizadas todas las páginas. El segundo nivel es cada

una de las plantillas específicas para cada acción (listar, editar o añadir y detalles) y estructura del proyecto.

### 2.1.6. WebStorm

Se ha usado WebStorm como entorno de desarrollo. Este software está especialmente diseñado para desarrollar aplicaciones web con Angular2 entre otros (en este proyecto ha sido el framework usado). Permite configurar el entorno de trabajo según las preferencias del usuario, esto hace más cómodo trabajar con él.

Se basa en plugins, es decir, se pueden instalar plugins que autocompletan y marcan errores a medida que se va escribiendo el código, esto permite programar de forma más eficiente y rápida. No solo reconoce Angular2, si quieres trabajar con algún otro lenguaje o framework (como Twig) únicamente se debe buscar en el gestor de plugins incluido e instalarlo. A parte de los plugins que facilitan la escritura de código también existen para personalizar el entorno o de integración con otras herramientas como Git.

### 2.1.7. Git

Para el control de versiones se ha usado Git. Git permite la gestión de proyectos mediante diferentes ramas. Al estar únicamente una persona trabajando en el proyecto, se podría haber trabajado sobre la rama "master" directamente (es la que se crea automáticamente), pero se ha preferido trabajar con diferentes ramas adaptando este proyecto a los otros proyectos de la empresa como se explica a continuación.

A parte de la rama "master" se han creado las ramas "development" y "developmentIrene". La rama "master" no se ha tocado durante el transcurso del proyecto, a la rama development se han subido los cambios significativos (funcionalidades que ya estaban listas). La rama "developmentIrene", ha sido la más usada, donde se han ido subiendo cambios constantemente por más pequeños que fueran.

A pesar de que hay un gran número de software que permite usar Git con una interfaz de usuario (como TreeSource o plugins de WebStorm), se ha preferido usar mediante la línea de comandos debido a su flexibilidad y rapidez. Se ha creado el repositorio en el servidor Bitbucket.

### 2.1.8. Guiffy

Guiffy es un software que permite juntar dos ficheros en uno mediante uniones a dos y tres vías. La unión (merge) a tres vías consiste en juntar los dos ficheros que se le pasan, en un tercer fichero que contiene la información de ambos. La unión a dos vías es igual que el anterior pero en vez de crear un nuevo fichero, modifica el primer fichero que se le pasa. A la hora de juntar los dos ficheros se le debe indicar qué cambios tienen que permanecer y cuáles se descartan mediante una interfaz gráfica.

Puede ser usado mediante una API por línea de comandos o con su interfaz. Pero aunque sea usado por la línea de comandos, al hacer la unión se le deben indicar los cambios mediante la interfaz como ya se ha comentado. La dificultad de automatización del proceso, ha sido el factor por el que se ha descartado su uso.

## 2.2. Situación inicial del proyecto

La empresa ha ido teniendo diferentes versiones de la aplicación desarrollada desde que se fundó. Estas versiones se han ido construyendo a medida que los lenguajes de programación, con los que estaba desarrollada, iban evolucionando y se iban quedando antiguos y sin mantenimiento por parte de las empresas dedicadas a ellos y comunidad de programadores.

En este caso, aunque ya existía la aplicación desarrollada, esta únicamente se ha usado como referencia y para obtener con mayor facilidad los requisitos del nuevo sistema. Por lo que la nueva aplicación se ha construido partiendo desde cero.

# Capítulo 3

# Planificación del proyecto

## 3.1. Metodología

El proyecto se ha llevado a cabo mediante una metodología ágil. Aunque al principio se pensó aplicar una metodología tradicional, debido al desconocimiento de gran parte de los lenguajes de programación y herramientas utilizadas, finalmente, se decidió aplicar una metodología basada en Scrum a la hora de generar automáticamente el administrador. Las metodologías ágiles se adaptan muy bien a los cambios sin poner en riesgo el resto de la planificación, como suele ocurrir en las metodologías tradicionales. Scrum se trata de una metodología en la que se realizan incrementos periódicos (denominados sprints), en la figura 3.1 se puede ver un gráfico de esta metodología.

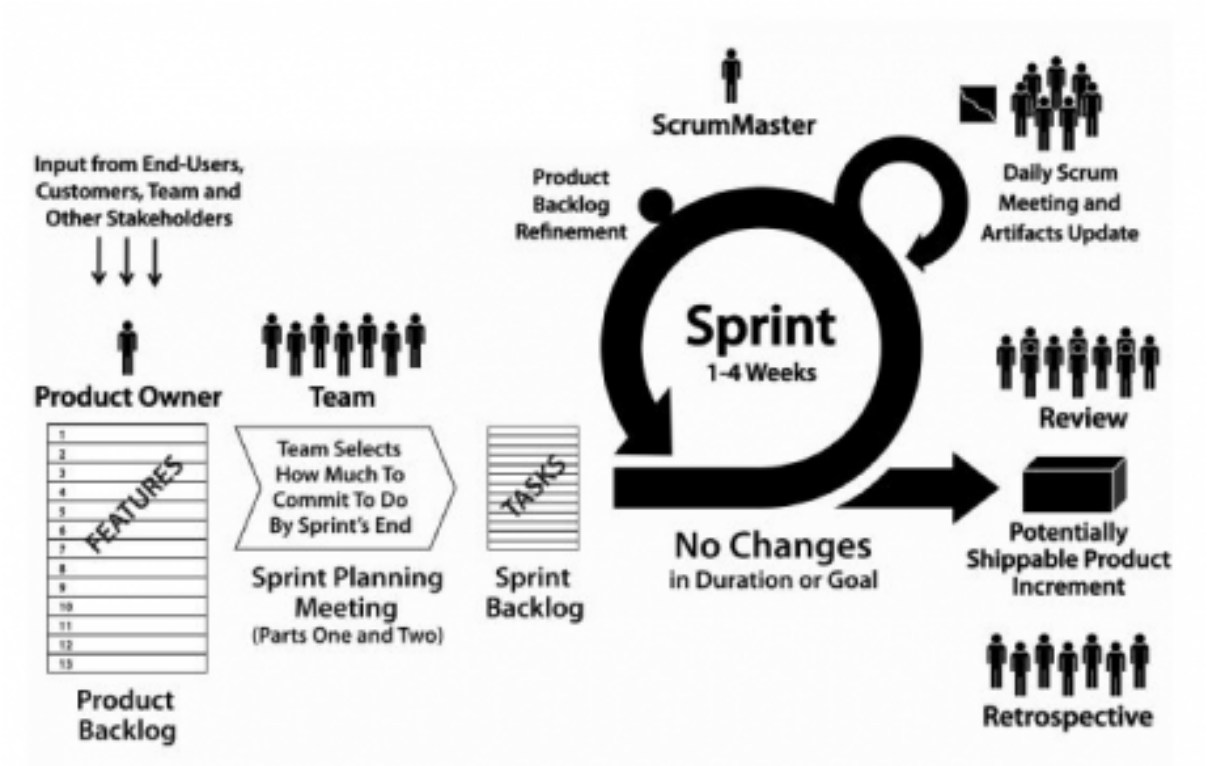


Figura 3.1: Gráfico sobre el ciclo de desarrollo de la metodología ágil basada en Scrum. [6]

El primer paso es obtener un listado de tareas y requisitos (en inglés product backlog). Este listado no está cerrado, a medida que se vaya desarrollando el proyecto se puede incrementar con tareas que no se habían tenido en cuenta al principio.

Una vez obtenido el listado de tareas y requisitos de la aplicación, hay que establecer los puntos de historia de cada uno de ellos. Los puntos de historia (story points) consisten en una estimación de lo que se cree que puede costar la realización de la tarea. Se trata de ordenar las tareas según su dificultad. Esto es importante a la hora de planificar los sprints, para no asumir demasiada carga de trabajo que luego no se podrá finalizar.

Se debe definir la duración de los sprints, el tiempo que se tiene para desarrollar un incremento al producto añadiéndole más funcionalidades. En cada sprint se tendrá de elegir una pila de tareas a realizar (sprint backlog). Para elegir las tareas se debe mirar el número de puntos de historia asignados así como la importancia de las mismas. Es posible que durante el sprint no se puedan terminar todas las tareas elegidas debido a una carga de trabajo no contemplada. En dicho caso, se deben tener en cuenta en el siguiente sprint y reajustar el número de puntos de historia a realizar.

Durante cada sprint se realizan reuniones diarias (dally scrum meeting) para poner en común todo lo avanzado en el proyecto durante el día anterior. Y al final de cada sprint se realiza una reunión con todos los implicados en el proyecto durante la cual se prueba el producto obtenido (se realiza una demo) y se proponen correcciones y mejoras

La recopilación de requisitos se ha llevado a cabo mediante reuniones previas con el responsable del desarrollo del proyecto. Como la aplicación desarrollada va a ser usada por todos los miembros del equipo de desarrollo de la empresa, también se han llevado a cabo entrevistas con parte del equipo. Mientras que la reunión se ha hecho en un despacho, las entrevistas se han realizado de forma más informal en la cafetería durante las horas de descanso. Los usuarios han hecho comentarios acerca de comportamientos que creían que serían útiles para el futuro sistema, luego se han comentado estas mejoras con el responsable del proyecto, para que las aprobara o rechazara.

También se ha hecho un diseño de la interfaz gráfica. Para el proyecto, este es uno de los puntos cruciales. El diseño de la interfaz gráfica se ha empezado haciendo con diferentes mockups a mano, con papel y lápiz, para luego pasarlos al ordenador mediante el programa "Balsamiq mockup" y mostrarlos al responsable quien los validó.

En este proyecto se ha establecido una duración del sprint de una semana. Las reuniones diarias se han hecho a primera hora del día, para ello se reunían el responsable de la base de datos y la estudiante. Al final de cada sprint se han realizado reuniones con el equipo (futuro usuario de la aplicación) y responsable del proyecto, para validar y verificar el trabajo realizado y proponer mejoras.

## 3.2. Planificación

Para la planificación del proyecto se ha hecho un reparto de todas las horas de las que se dispone, sumando un total de 435 horas. Dentro de las 135 horas que se deben hacer fuera de la empresa, se ha incluido las partes de "desarrollo de la propuesta técnica", "definición de requisitos, análisis y diseño" y "documentación". Todas las otras partes están dentro del trabajo a desarrollar en la empresa. Hay tareas que se realizan de forma paralela, en la empresa y en casa.

El desarrollo del proyecto se ha dividido en tres partes muy bien diferenciadas. La primera fase es de investigación y estudio de los diferentes lenguajes de programación usados en el transcurso del proyecto. La segunda fase es de desarrollo de un administrador de forma manual para estar seguro que se ha entendido lo que es un administrador y validar los requisitos de la aplicación web. Y en la última fase se automatiza el desarrollo de los administradores. Estas tres partes se han implementado de forma secuencial y en el orden descrito, es dentro de la última fase donde se ha seguido la metodología ágil basada en Scrum.

En la figura 3.2 se muestran con detalle el listado de las tareas planificadas para la estancias en prácticas, con las horas dedicadas a cada una y las fechas previstas. A continuación se muestra el mismo listado únicamente con el título de las tareas y subtareas:

- Inicio
  - Definir el proyecto con el tutor y supervisor
  - Definir el modo de trabajo
- Documentar y planificar

    - Contexto y búsqueda de información
    - Identificación del alcance y funcionalidades específicas
- Planificar
    - Detallar las tareas y estimar costes
    - Redactar de la propuesta técnica
    - Entregar de la propuesta
- Definir requisitos, análisis y diseño
    - Diagrama de casos de uso
    - Diagrama de clases
    - Diseño de interfaces
    - Validar el diseño de las interfaces de usuario
- Desarrollo técnico y formación
    - Formación
        - Estudiar Angular 2
        - Estudiar Bootstrap
        - Estudiar PHP
        - Estudiar Twig
    - Desarrollo del producto
        - Desarrollo del administrado de forma manual
            - Menú y navegación
            - Listado y detalle
            - Formularios
            - Comunicación cliente-servidor
        - Generación de código automática
- Documentación
    - Redacción de informe quincenal 1
    - Redacción de informe quincenal 2
    - Redacción de informe quincenal 3
    - Redacción de informe quincenal 4
    - Redacción de informe quincenal 5
    - Redacción de informe quincenal 6
- Redacción de la memoria técnica
- Entrega de la memoria
- Preparación de la presentación oral
- Presentación oral

La tarea de formación se ha reducido bastante por el hecho que únicamente se ha seguido un curso online durante dos semanas en las que se ha aprendido los conceptos básicos de Angular2. El estudio de Bootstrap, PHP y Twig se ha hecho a medida que se ha ido desarrollando el código, por lo que esta primera parte se ha finalizado antes de lo contemplado. Las horas sobrantes se han añadido a la generación de código automática. Por lo que dicha tarea se ha terminado el 7/11/2016 e inmediatamente se ha empezado con la siguiente.

El desarrollo manual del administrador no se ha completado. Se han desarrollado dos entidades completamente diferentes para comprobar que el código es muy parecido, prácticamente el mismo. Se ha empleado el tiempo establecido y al finalizar se ha pasado directamente a la última parte del proyecto, la generación de código.

En el desglose de tareas anterior se ha dejado la tarea de "Generación automática" sin indicar ninguna subtarea. Como se ha indicado anteriormente, aquí se ha aplicado una metodología basada en Scrum, se ha empezado el 19/12/2016 y se ha empleado todo el tiempo restante en ella hasta el 20/01/2017. Se pensó realizar cuatro sprints y dejar la última semana para mejorar el aspecto visual de la aplicación, finalmente se han empleado los cinco sprints para la implementación de toda la funcionalidad de la aplicación web. En el "Capítulo 5 Implementación y prueba" se detallan cada uno de los sprints.

### 3.2.1. Historias de usuario

Las historias de usuario permiten a los clientes y usuarios finales de la aplicación, indicar de forma sencilla y rápida (en una sola frase en lenguaje natural) qué es lo que esperan del sistema. Esta forma de recopilar requisitos está hecha para que cualquier persona, con o sin conocimientos de informática, sea capaz de visualizar y expresar qué tipo de sistema quiere.

- HU_ADM_01: Como usuario quiero poder listar todas las entidades en tablas (datatables).
- HU_ADM_02: Como usuario quiero poder modificar las entidades.
- HU_ADM_03: Como usuario quiero poder ver los detalles de la entidad elegida.
- HU_ADM_04: Como usuario quiero poder eliminar las entidades que no me interesen.
- HU_ADM_05: Como usuario quiero poder buscar entidades por cualquier campo.
- HU_ADM_06: Como usuario quiero poder crear entidades en la base de datos.
- HU_ADM_07: Como usuario quiero poder navegar de forma cómoda entre las páginas del administrador.
- HU_ADM_08: Como usuario quiero que la aplicación tenga un estilo definido.
- HU_GEN_01: Como usuario quiero que el sistema sea capaz de leer los datos del fichero de entrada.
- HU_GEN_02: Como usuario quiero que el sistema sea capaz de generar un administrador web que gestione una base de datos.

## 3.3. Estimación de recursos y costes del proyecto

En este apartado se detallan todos los recursos (con importe económico) de los que se hizo uso durante el transcurso de la estancia en prácticas. En el cuadro 3.1 se muestran los importes y a continuación se justifican:

- En la oficina se han trabajado trescientas horas en total, durante los tres meses de prácticas. Se ha supuesto que cada mes la empresa paga a sus trabajadores un salario de 1.200 euros en una jornada normal de 8 diarias y que un mes tiene 30 días.
- Se ha supuesto que el ordenador portátil proporcionado por la empresa cuesta actualmente en el mercado 600 euros, tiene una vida útil de 5 años y se usó durante un periodo de tres meses.
- Se usó la plantilla "Sing App" para Angular2 que se compró el 19/12/2016.
- Se siguió un curso Angular2 comprado y realizado en la página de udemy, con título "Angular2 the complete guide".
- Se utilizó el software de Jira para el seguimiento del trabajo y control de las horas realizadas.

| **Concepto** | **Unidades** | **Precio unitario** | **Total** |
|---|---|---|---|
| Horas trabajadas | 300 horas | 5 €/hora | 1.500 € |
| Ordenador portátil | 3 meses | 120 €/año | 30 € |
| Licencia WebStorm | 1 licencia | 649 €/año | 162,25 € |
| Proyecto Bitbucket privado | 1 proyecto | 2 €/mes | 6 € |
| Plantilla HTML | 1 plantilla | 85 € | 85 € |
| Curso Angular2 | 1 curso | 190 € | 190 € |
| Sesión Jira | 1 sesión | 6 €/mes | 18 € |
| | | **TOTAL:** | 1.991,25 € |

Cuadro 3.1: Desglose de recursos y costes de la estancia en prácticas.

Todos los importes que se pagan por meses o se pueden descomponer en meses, se han calculado con respecto a los meses durante los que se realizó la estancia en prácticas. Todos los precios se han tomado en función a lo que valían en el momento de la compra, por lo que es posible que en el momento actual se hayan modificado decrementándose o incrementándose.

## 3.4. Seguimiento del proyecto

El seguimiento del proyecto se ha hecho diariamente mediante la plataforma Jira, en la que han creado etiquetas para cada tarea y luego se indicaban las horas trabajadas en dicha tarea. Cuando se terminaba una tarea se cambiaba su estado, de la misma manera cuando se reabría.

Por otra parte, diariamente me reunía con los compañeros y les mostraba lo que llevaba hecho para que ellos me dieran consejos sobre cómo mejorarlo y para coordinarme con el compañero

que se encargaba de la base de datos. Además semanalmente se mostraba el trabajo realizado al responsable para que diera su visto bueno.

| | | Mod de | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|---|---|
| 1 | | | **Proyecto** | **400 horas** | **lun 24/10/16** | **vie 17/02/17** | |
| 2 | | | **Inicio** | **3 horas** | **lun 24/10/16** | **lun 24/10/16** | |
| 3 | | | Definir el proyecto con el tutor y supervisor | 2 horas | lun 24/10/16 | lun 24/10/16 | |
| 4 | | | Definir el modo de trabajo | 1 hora | lun 24/10/16 | lun 24/10/16 | 3 |
| 5 | | | **Documentar y planificar** | **2 horas** | **lun 24/10/16** | **lun 24/10/16** | |
| 6 | | | Contexto y búsqueda de información | 1 hora | lun 24/10/16 | lun 24/10/16 | 4 |
| 7 | | | Identificación del alcance y funcionalidades específicas | 1 hora | lun 24/10/16 | lun 24/10/16 | 6 |
| 8 | | | **Planificación** | **17 horas** | **mar 25/10/16** | **vie 28/10/16** | |
| 9 | | | Detallar las tareas y estimar costes | 2 horas | mar 25/10/16 | mar 25/10/16 | 7 |
| 10 | | | Redacción de la propuesta técnica | 15 horas | mar 25/10/16 | vie 28/10/16 | 9;7;6 |
| 11 | | | Entrega de la propuesta | 0 horas | vie 28/10/16 | vie 28/10/16 | 10 |
| 12 | | | **Definir requisitos análisis y diseño** | **8 horas** | **vie 28/10/16** | **lun 31/10/16** | |
| 13 | | | Diagrama de casos de uso | 2 horas | vie 28/10/16 | vie 28/10/16 | 11 |
| 14 | | | Diagrama de clases | 2 horas | vie 28/10/16 | lun 31/10/16 | 11;13 |
| 15 | | | Diseño de las interfaces de usuario | 4 horas | lun 31/10/16 | lun 31/10/16 | 11;13;14 |
| 16 | | | Validar diseño | 0 horas | lun 31/10/16 | lun 31/10/16 | 15 |
| 17 | | | **Desarrollo técnico y formación** | **300 horas** | **lun 24/10/16** | **vie 20/01/17** | |
| 18 | | | **Formación** | **100 horas** | **lun 24/10/16** | **lun 21/11/16** | |
| 19 | | | Estudio del lenguaje de programación AngujarJS 2 | 50 horas | lun 24/10/16 | lun 07/11/16 | |
| 20 | | | Estudio del lenguaje de programación Bootstrap | 25 horas | mar 08/11/16 | lun 14/11/16 | 19 |
| 21 | | | Estudio del lenguaje de programación PHP | 10 horas | mar 15/11/16 | mié 16/11/16 | 20 |
| 22 | | | Estudio del lenguaje de programación Twig | 15 horas | jue 17/11/16 | lun 21/11/16 | 21 |
| 23 | | | **Desarrollo del producto** | **200 horas** | **mar 22/11/16** | **vie 20/01/17** | |
| 24 | | | **Desarrollo del administrador de forma manual** | **120 horas** | **mar 22/11/16** | **mié 28/12/16** | |
| 25 | | | Menú y navegación | 15 horas | mar 22/11/16 | jue 24/11/16 | 22 |
| 26 | | | Listado y detalle | 20 horas | vie 25/11/16 | mié 30/11/16 | 25 |
| 27 | | | Formularios | 40 horas | jue 01/12/16 | mié 14/12/16 | 26 |
| 28 | | | Comunicación cliente-servidor | 45 horas | jue 15/12/16 | mié 28/12/16 | 27 |
| 29 | | | Generación de código automática | 80 horas | jue 29/12/16 | vie 20/01/17 | 21;22;25;26;27;28 |
| 30 | | | **Documentación** | **305 horas** | **lun 21/11/16** | **vie 17/02/17** | |
| 31 | | | **Redacción de informes quincenales** | **181 horas** | **lun 21/11/16** | **lun 16/01/17** | |
| 32 | | | Redacción de informes quincenales 1 | 1 hora | lun 21/11/16 | lun 21/11/16 | |
| 33 | | | Redacción de informes quincenales 2 | 1 hora | lun 05/12/16 | lun 05/12/16 | |
| 34 | | | Redacción de informes quincenales 3 | 1 hora | lun 19/12/16 | lun 19/12/16 | |
| 35 | | | Redacción de informes quincenales 4 | 1 hora | lun 02/01/17 | lun 02/01/17 | |
| 36 | | | Redacción de informes quincenales 5 | 1 hora | lun 16/01/17 | lun 16/01/17 | |
| 37 | | | Redacción de la memoria técnica | 80 horas | lun 23/01/17 | lun 13/02/17 | 29;16 |
| 38 | | | Entrega de la memoria | 0 horas | lun 13/02/17 | lun 13/02/17 | 37 |
| 39 | | | Preparación de la presentación oral | 20 horas | mar 14/02/17 | vie 17/02/17 | 38 |
| 40 | | | Presentación oral | 0 horas | vie 17/02/17 | vie 17/02/17 | 39 |

Figura 3.2: Desglose de tareas, junto con la duración de las mismas y sus precedentes.

# Capítulo 4

# Análisis y diseño del sistema

## 4.1. Análisis del sistema

En esta sección se muestran los requisitos, diagrama de casos de uso y diseño de las diferentes partes de la aplicación. Debido a que se trata de un generador, parte de los diagramas (el diagrama de clases o de actividades) vienen dados por el fichero XML que contiene la estructura de datos que el usuario define. Todos los diagramas que dependen de dicho fichero, se muestran con un ejemplo concreto.

Al tratarse de una aplicación web que se ejecuta en el navegador, se ha de tener en cuenta la navegación por medio de la ruta URL. El usuario puede acceder a entidades no persistidas en la base de datos, en este caso se le redirigirá a una página de error 404.

### 4.1.1. Requisitos

Los requisitos permiten identificar las funcionalidades del sistema informático y qué es lo que esperan los usuarios de la aplicación. Se han establecido tres tipos de requisitos, los funcionales, de hardware y software y de calidad. Los requisitos funcionales describen el comportamiento la aplicación, es decir, qué es lo que se espera que haga. Los requisitos de hardware y software indican el comportamiento del sistema frente a factores ajenos a la aplicación, es decir, las características mínimas sobre las que la aplicación debe funcionar correctamente. Y los requisitos de calidad permiten un código que se adapte a los estándares de la empresa.

A continuación se relacionan los requisitos tanto del administrador como del generador. Los requisitos funcionales que se muestran a continuación son adicionales a las historias de usuario, se han especificado desde el punto de mejorar la GUI y con ello la experiencia de usuario. En los siguientes subapartados se explican con más detalle las funcionalidades del sistema:

- Requisitos de hardware y software.
    - El sistema de debe ser reactivo (responsive), es decir, se debe adaptar a todo tamaño

de pantalla.

- El sistema se debe poder usar en los navegadores actuales: Chrome, Microsoft Edge o Mozilla Firefox.

- Requisitos de calidad.
  - Código modular y encapsulado para más fácil depuración.
  - Un diseño de la API Rest adaptada a estándares.
  - Reducir la carga memorística de los usuarios.

- Requisitos funcionales.
  - El sistema debe partir de un "ng new my-app" que creará la estructura básica de ficheros, a partir de ahí se deben crear todos los elementos necesarios y modificar los ya existentes si se requiere.
  - El sistema debe dar soporte de los tipos de datos básicos.
  - El sistema ha de tener una opción de generación de mapas (con la API de google maps).
  - El sistema debe proporcionar paginación en todos los listados.
  - El sistema debe proporcionar validación de formularios en el lado del cliente.
  - El sistema ha de mostrar confirmaciones al eliminar elementos.
  - El sistema debe contemplar la creación de un menú de "migas" (breadcrumb) para todas las páginas.

### 4.1.2. Diagrama de casos de uso

El diagrama de casos de uso sirve para mostrar de forma gráfica como un actor interactúa con el sistema. Los actores representan cada uno de los roles con los que los usuarios operan con el sistema, los casos de uso son las operaciones que se deben poder ejecutar en la aplicación y las relaciones representan las interacciones entre los actores y los casos de uso.

En el diagrama de casos de uso mostrado en la figura 4.1, se ha incluido tanto los casos de uso del administrador como los del generador. El generador genera el administrador a partir de los datos leídos del fichero XML, el administrador cuenta con las funcionalidades CRUD y la de buscar.

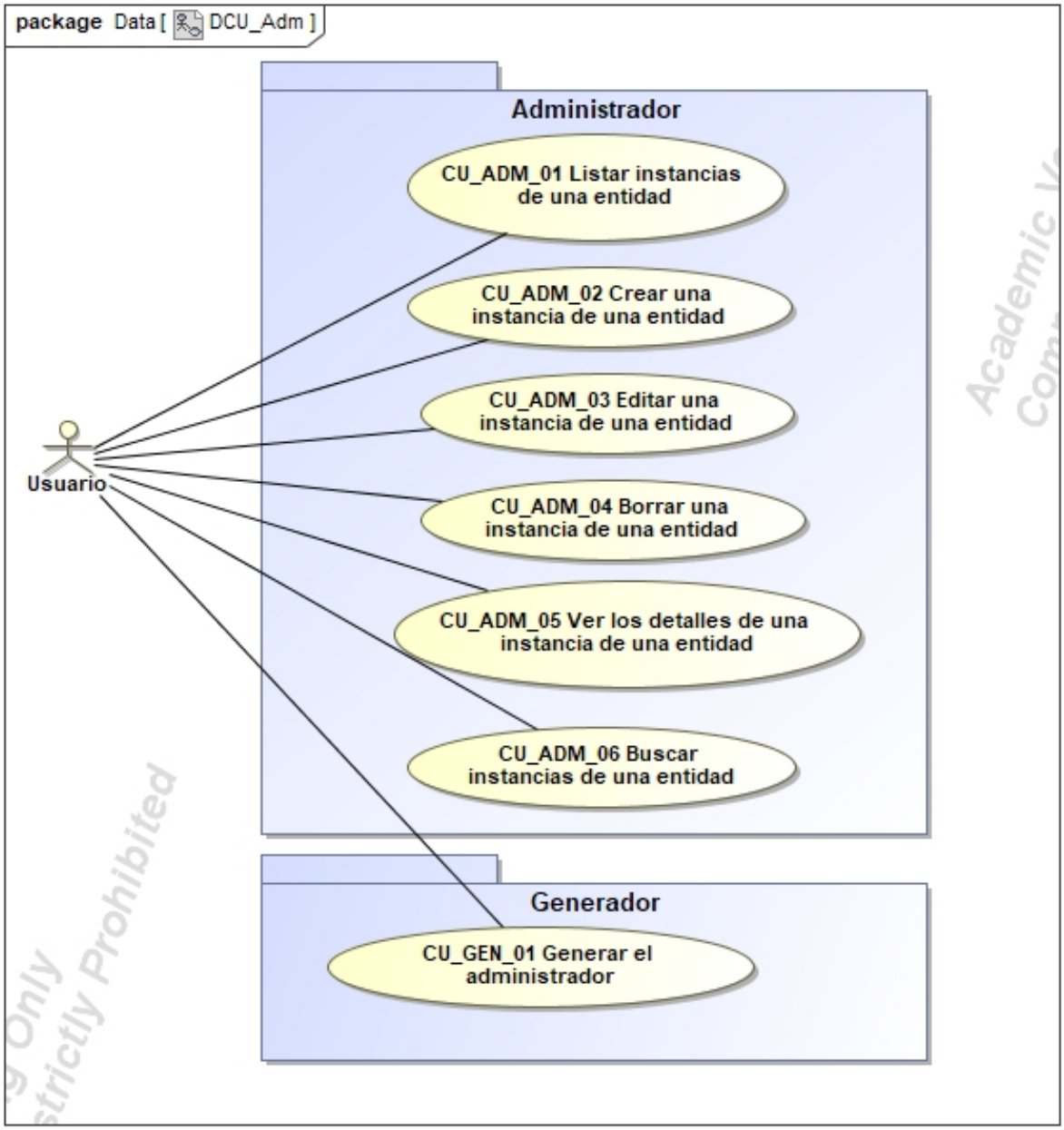


Figura 4.1: Diagrama de casos de uso de la aplicación.

### 4.1.3. Especificación de los casos de uso

En las tablas 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 y 4.7 se muestran las especificaciones de todos los casos de uso relacionados en el diagrama de casos de uso anterior 4.1.

| | |
|---|---|
| Identificador | CU_ADM_01 |
| Nombre | Listar instancias de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario la visualización de un subconjunto de atributos de las instancias de las entidades almacenadas. |
| Alcance | La visualización en una tabla de las instancias persistidas en la entidad. |
| Nivel | Tarea principal |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias de usuario relacionadas | HU_ADM_01 |
| Precondición | Debe haber al menos una entidad creada y almacenada en la base de datos para que se liste (el administrador debe estar generado). |
| Condición fin con éxito | Que el usuario haya seleccionado un tipo de entidad para que se listen los atributos de todas las instancias de la entidad. |
| Condición fin con fracaso | Que no haya ninguna instancia para listar en la base de datos o no haya indicado ninguno de sus atributos para listar |
| Trigger | Seleccionar la entidad a listar. |
| Secuencia normal | Acciones |
| 1 | Obtener los datos de la base de datos |
| 2 | Mostrar los datos en una tabla. |
| Excepciones | - |
| Frecuencia esperada | Diariamente, varias veces por día |
| Importancia | Muy importante |
| Prioridad | Corto plazo |
| Comentarios | El listado debe hacerse mediante un “datatable” con paginación, para que no pida de una vez todos los datos a la base de datos y tarde demasiado en cargar. Debe ser rápido y fluido moverse entre las páginas del listado para una mejor experiencia de usuario. |

Cuadro 4.1: Especificación CU_ADM_01.

| | |
|---|---|
| Identificador | CU_ADM_02 |
| Nombre | Crear una instancia de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario añadir nuevas instancias a la base de datos. |
| Alcance | La visualización de un formulario con todos los campos vacíos y la opción de guardado. |
| Nivel | Tarea principal |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias de usuario relacionadas | HU_ADM_06 |
| Precondición | Seleccionar la entidad que se quiere almacenar en la base de datos (el administrador debe estar generado). |
| Condición fin con éxito | Que el usuario haya rellenado el formulario y lo haya guardado. |
| Condición fin con fracaso | Que el usuario se haya dejado algún campo obligatorio por rellenar o haya introducido de forma incorrecta algún campo. |
| Trigger | Seleccionar la opción de crear un nuevo elemento. |
| Secuencia normal | Acciones |
| 1 | Se muestra el formulario de creación de una instancia. |
| 2 | Rellenar el formulario |
| 2.1 | En caso de que la entidad se haya configurado para que acepte mapas. Seleccionar la opción mapa para añadir coordenadas clicando en el mapa. |
| 2.2 | En caso de que la entidad se haya configurado para que acepte traducciones. Seleccionar el lenguaje e introducir la traducción. |
| 3 | Seleccionar la opción de guardar. |
| 3.1 | Validar el formulario comprobando que todos los datos estén correctos. |
| 5 | Almacenar la información de la nueva instancia en la base de datos. |
| Excepción acción 3.1 | Excepción |
| 1 | El usuario selecciona la opción de guardar cuando no ha rellenado el formulario correctamente. En este caso se le señalizarán los campos que faltan por rellenar o no están correctos. La entidad no se guardará hasta que el formulario se rellene satisfactoriamente. |
| 2 | El usuario selecciona la opción de cancelar. En este caso no se guardará la entidad. |
| Frecuencia esperada | Semanalmente |
| Importancia | Muy importante |
| Prioridad | Corto plazo |

| | |
|---|---|
| Comentarios | En caso de que la entidad permita mapas, su creación se hará mediante un formulario y un mapa separados en dos pestañas diferentes. |

Cuadro 4.2: Especificación CU_ADM_02.

| | |
|---|---|
| Identificador | CU_ADM_03 |
| Nombre | Editar una instancia de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario seleccionar y modificar una instancia concreta para persistirla en la base de datos. |
| Alcance | La visualización de un formulario con sus campos rellenados para poder modificarlos y la opción de guardado (el administrador debe estar generado). |
| Nivel | Tarea principal |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias de usuario relacionadas | HU_ADM_02 |
| Precondición | El usuario debe seleccionar la entidad y la instancia a modificar. |
| Condición fin con éxito | Que el usuario haya modificado los campos del formulario y lo haya guardado. |
| Condición fin con fracaso | Que el usuario se haya dejado algún campo obligatorio por rellenar o haya introducido de forma incorrecta algún campo. |
| Trigger | Seleccionar la opción de modificar la instancia. |
| Secuencia normal | Acciones |
| 1 | Recuperar la instancia que se quiere modificar de la base de datos y mostrar por pantalla un formulario con sus datos. |
| 2 | Modificar los campos deseados del formulario. |
| 2.1 | Si acepta mapas, seleccionar la opción del mapa y añadir coordenadas clicando en el mapa o borrarlas seleccionando la opción de eliminar las coordenadas. |
| 2.2 | Si acepta traducciones, seleccionar la opción de las traducciones y añadirlas rellenando un campo con el nombre de la traducción y seleccionando su idioma o borrarlas seleccionando la opción de eliminar. |
| 3 | Seleccionar la opción de guardar. |
| 3.1 | Validar el formulario comprobando que todos los datos estén correctos. |
| 4 | Se guarda la instancia modificada en la base de datos. |
| Excepción acción 3.1 | Excepción |
| 1 | El usuario selecciona la opción de guardar cuando no haya rellenado el formulario correctamente. En este caso se le mostrarán los campos que faltan por rellenar o no están correctos. La entidad no se guardará hasta que el formulario se rellene satisfactoriamente. |
| 2 | El usuario selecciona la opción de cancelar. En este caso no se guardará la entidad. |
| Frecuencia esperada | Semanalmente |

| | |
|---|---|
| Importancia | Importante |
| Prioridad | Medio plazo |
| Comentarios | - |

Cuadro 4.3: Especificación CU_ADM_03.

| | |
|---|---|
| Identificador | CU_ADM_04 |
| Nombre | Borrar una instancia de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario seleccionar y eliminar una instancia de la base de datos. |
| Alcance | Opción de eliminado. |
| Nivel | Tarea principal |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias de usuario relacionadas | HU_ADM_04 |
| Precondición | El usuario debe seleccionar la entidad de la instancia a borrar (el administrador debe estar generado). |
| Condición fin con éxito | Que el usuario seleccione la opción de aceptar. |
| Condición fin con fracaso | Que el usuario seleccione la opción de cancelar. |
| Trigger | Seleccionar la opción de eliminar. |
| Secuencia normal | Acciones |
| 1 | Se muestra una venta de confirmación. |
| 2 | Confirmar la operación seleccionando la opción de aceptar. |
| 3 | La instancia se elimina de la base de datos. |
| Excepción acción 2 | Excepción |
| 1 | El usuario selecciona la opción de cancelar. En este caso no se guardará la entidad. |
| Frecuencia esperada | Esporádica |
| Importancia | Importante |
| Prioridad | Medio plazo |
| Comentarios | A la hora de borrar una instancia, debe aparecer una ventana modal con la confirmación de la operación, advirtiendo al usuario de la operación que se va a realizar. |

Cuadro 4.4: Especificación CU_ADM_04.

| | |
|---|---|
| Identificador | CU_ADM_05 |
| Nombre | Ver los detalles de una instancia de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario seleccionar y visualizar todos los atributos de una instancia de la base de datos. |
| Alcance | Listado de todos los atributos de la instancia |
| Nivel | Tarea secundaria |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias de usuario relacionadas | HU_ADM_03 |
| Precondición | El usuario debe seleccionar la entidad de la instancia que se quiere visualizar (el administrador debe estar generado). |
| Condición fin con éxito | Que el usuario seleccione la opción detalles de la instancia. |
| Condición fin con fracaso | - |
| Trigger | Seleccionar la opción de mostrar la instancia. |
| Secuencia normal | Acciones |
| 1 | Recuperar de la base de datos los atributos de la instancia seleccionada. |
| 2 | Mostrar por pantalla la información de la instancia |
| Excepción | - |
| Frecuencia esperada | Esporádica |
| Importancia | Importante |
| Prioridad | Largo plazo |
| Comentarios | A la hora de mostrar los detalles de una instancia, se debe de hacer de forma amigable para que el usuario pueda entender toda la información recogida en esta pantalla de un solo vistazo. |

Cuadro 4.5: Especificación CU_ADM_05.

| | |
|---|---|
| Identificador | CU_ADM_06 |
| Nombre | Buscar instancias de una entidad |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe permitir al usuario encontrar una instancia almacenada en la base de datos. |
| Alcance | Encontrar y mostrar todas las instancias que contengan la cadena de caracteres especificada por el usuario en al menos uno de sus atributos. |
| Nivel | Tarea secundaria |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias relacionadas | HU_ADM_05 |
| Precondición | El usuario debe seleccionar la entidad de la instancia que se quiere buscar (el administrador debe estar generado). |
| Condición fin con éxito | Que el usuario introduzca una cadena que coincida con alguna instancia. |
| Condición fin con fracaso | Que la cadena indicada no coincida con ninguna instancia. |
| Trigger | Al introducir el texto que se quiere buscar. |
| Secuencia normal | Acciones |
| 1 | Recuperar las instancias de la entidad que correspondan con la búsqueda de la base de datos. |
| 2 | Mostrar las instancias ordenadas en una tabla. |
| Excepciones | - |
| Frecuencia esperada | Frecuente |
| Importancia | Importante |
| Prioridad | Largo plazo |
| Comentarios | Las búsquedas se harán en toda la base de datos, no solo en la página actual de la tabla. Por lo que si la tabla está formada por varias páginas, el elemento debe ser mostrado independientemente de la página en la que se encuentre. Debe existir un campo de búsqueda por cada atributo de la instancia que se pueda buscar. |

Cuadro 4.6: Especificación CU_ADM_06.

| | |
|---|---|
| Identificador | CU_GEN_01 |
| Nombre | Generar el administrador |
| Versión | 1 |
| Autores | Irene Roures Villalonga |
| Fuentes | Responsable del proyecto |
| Descripción | El sistema debe obtener todos los datos necesarios de la lectura del fichero XML que el usuario le indique y a partir de ellos crear el administrador. |
| Alcance | Obtener la información relevante para la generación del administrador y generar la estructura de ficheros y el código para dotar de funcionalidad al administrador |
| Nivel | Tarea principal |
| Actor principal | Usuario |
| Actores secundarios | - |
| Relaciones | - |
| Historias relacionadas | HU_GEN_01 y HU_GEN_02 |
| Precondición | Situarse en la carpeta "generator" del proyecto. |
| Condición fin con éxito | Que el usuario haya creado correctamente el fichero XML y lo haya introducido en la carpeta "generator". |
| Condición fin con fracaso | Que el fichero XML no exista en la carpeta correcta o sea incorrecto. |
| Trigger | Ejecutar el fichero generador con el comando: `php main-generator.php`. |
| Secuencia normal | Acciones |
| 1 | Leer los datos de los elementos `table` y `column`. |
| 2 | Convertir los tipos de los elementos `column` |
| 3 | Leer los datos de los elementos `relation`. |
| 4 | Para todas las relaciones de la entidad que se está generando, si es la parte uno de la relación añadir un campo en todos los formularios. |
| 5 | Si tiene traducciones, crear una clase que las contenga y añadirlas en las páginas de editar, crear y detalles y crear una clase que contenga las traducciones. |
| 6 | Si tiene mapas, incrustar el mapa de google maps en las páginas de editar, crear y detalles de la entidad y crear una clase que contenga las marcas para el mapa. |
| 7 | Generar los ficheros de listar, crear y editar, detalles, la clase propia de la entidad con todos sus atributos, el módulo y el servicio. |
| Excepciones | - |
| Frecuencia esperada | Esporádica |
| Importancia | Muy importante |
| Prioridad | Corto plazo |

| | |
|---|---|
| Comentarios | Siempre se partirá con un proyecto Angular2 creado mediante la instrucción: `ng new administrador` que crea la estructura base y en el directorio `/src` se le tiene que añadir la carpeta "generator" con el generador. Las traducciones estarán disponibles en español (es), alemán (de), francés (fr) e inglés (en). |

Cuadro 4.7: Especificación CU_GEN_01.

## 4.2. Diseño del sistema

El sistema está formado por dos proyectos, el que contiene todos los ficheros PHP y plantillas Twig necesarias para la generación automática del administrador y el proyecto Angular2 del administrador con todas sus funcionalidades específicas.

La aplicación desarrollada está formada por tres partes diferentes como se muestra en la figura 4.2, lectura del fichero XML, generación automática del administrador y generación automática de la base de datos. Lo primero que se hace es la lectura de un fichero en formato XML, que es el único dato de entrada. Este fichero es lo más importante para una correcta generación del administrador que el usuario verá por pantalla, mediante los datos introducidos en el fichero se genera automáticamente toda la aplicación web además de la base de datos.

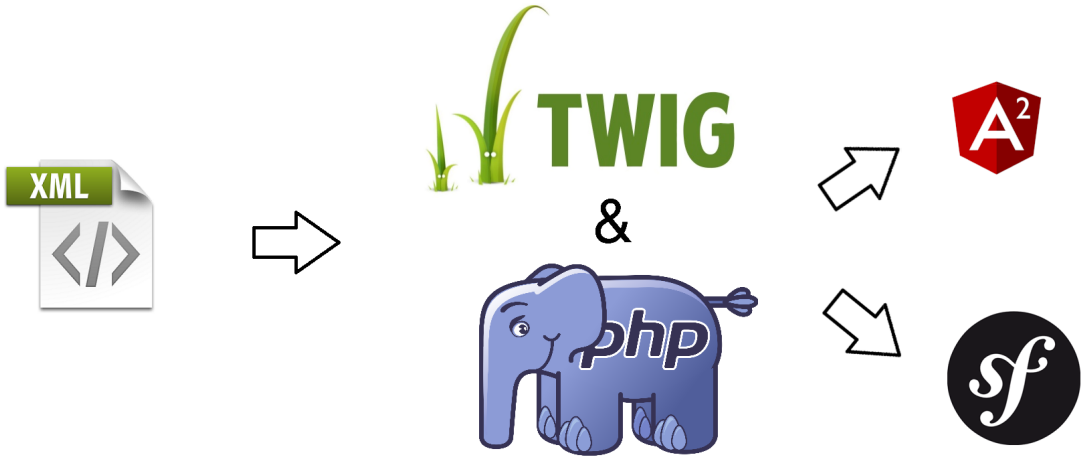


Figura 4.2: Estructura general de toda la aplicación.

### 4.2.1. Diseño de la estructura del fichero XML para la generación del administrador

Un fichero XML puede tener cualquier estructura. En este caso, el fichero XML empieza declarando una aplicación dentro de la cual se declara la base de datos con las tablas y las relaciones con columnas. Todos estos elementos tienen atributos (a los que se les puede dar un valor) que permiten indicar ciertas propiedades necesarias. El fichero XML tiene la estructura de un JSON con todos los datos necesarios para la generación del administrador (base de datos y GUI).

Para permitir la máxima personalización del administrador, se han diseñado una serie de atributos en el fichero XML. Algunos de estos atributos obligan a la creación de entidades auxiliares que permitan persistir sus datos. A continuación se explica de para qué sirven los más importantes para el generador y dónde aplicarlos.

Según la estructura del fichero XML, el primer atributo es el nombre de la aplicación. Aunque parece un dato irrelevante, es importante ya que el administrador tendrá el nombre que se haya establecido en el atributo. En el elemento `application` debemos añadir el atributo **name** con el nombre del administrador, como se muestra en la figura 4.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="file:/Users/uji/Documents/workspace/480MDF/schema
        /database.xsd"
    name="CRMInterno"
    package="com.cuatroochenta.eroskinavidad">
    ...
    ...
</application>
```

Figura 4.3: Ejemplo de la creación de una aplicación.

Dentro de la aplicación se empieza declarando la base de datos y dentro de esta todas las tablas y relaciones. Cada una de las tablas tiene un conjunto amplio de atributos personalizados, se pueden dividir en dos tipos: los que se introducen como atributos en la declaración de la tabla (y afectarán a toda la tabla) y los que se indican en las columnas de cada tabla (que solo modifican la columna sobre la que se añaden). A continuación se explican los atributos que se aplican a toda la tabla:

- Los atributos `label` y `pluralLabel` hacen referencia al nombre en singular y plural de la entidad que se quiere que aparezca en la interfaz.
- El atributo `icon` hace referencia al icono que se quiere que aparezca junto con el nombre de la entidad.
- El atributo `genre` indica el género de la entidad, usado para una mejor concordancia entre los artículos y el nombre de las entidades en las frases y títulos que aparecen en la interfaz.
- Los atributos `name` y `pluralName` son usados para la generación del código del administrador, indican lo mismo que los atributos `label` y `pluralLabel`.
- El atributo `i18n` se trata de las traducciones. Al añadir este atributo, la aplicación permite traducir el contenido del elemento `column` donde se haya introducido el atributo `translations` con valor verdadero. Por ejemplo en la tabla declarada en la figura 4.4 se traduce el elemento `column` cuyo atributo `translation` tiene valor `true`, no el atributo `color`. Están disponibles las traducciones al español, inglés, francés y alemán. Se permite una pequeña gestión de los lenguajes con las opciones de crear y eliminar.

```
<table name="Estado" pluralName="estados" i18n="true" icon="fa fa-info" sqlTableName="estado"
    label="Estado" pluralLabel="Estados" syncMode="none" generateSymfonyModule="true
    genre="male">
<column name="id" sqlColumnName="id" type="PKINT" primaryKey="true" autoIncrement="true"/>
<column name="color" type="VARCHAR" label="Color" color="true" listable="true" form="true"/>
<column name="nombre" translations="true" type="VARCHAR" size="255" label="Nombre"
    description="true" searchable="true" listable="true" form="true"/>
</table>
```

Figura 4.4: Ejemplo de una tabla con traducciones.

Las etiquetas que se indican en cada columna son las siguientes:

- El atributo `listable` indica que el contenido del elemento `column` debe aparecer en el listado de la entidad. Es obligatorio, su valor es verdadero (true) o falso (false) dependiendo de si queremos o no que se liste.
- El atributo `serchable` indica que se puede buscar una instancia por este atributo. Su funcionalidad es similar al atributo anterior. Normalmente irán a la par, pero en algunos casos puede interesar que el contenido del elemento se liste pero no que se pueda buscar por él.
- El atributo `form` sirve para saber qué atributos deben ser mostrados en el formulario de crear o editar. Es booleano por lo que funciona de la misma manera que los anteriores. Por defecto todos los elementos que se muestran en el formulario son campos de entrada (inputs) normales (de texto simple) a no ser que se especifique algún otro atributo.
- El atributo `email` sirve para saber que se trata de un correo electrónico, usado para la validación del formulario.
- El atributo `label`, al igual que con las entidades, sirve para que mostrar el nombre del input en los formularios
- El atributo `color`, indica que cierta entidad tiene que tener un elemento donde poder elegir un color ("color picker"). Si se indica, su valor debe ser verdadero.
- El atributo `dropdown`, es usado para los elementos que contienen relaciones. Dependiendo de si la relación es "uno a uno" o "muchos a uno", este campo permite elegir un elemento o un conjunto de elementos. Esto se hace automáticamente sin que se tenga que indicar nada.
- El atributo `short`, sirve para añadir al formulario casillas de verificación (checkboxes o radiobuttons). Si el valor de este atributo es falso, se añaden radiobuttons. En cambio, si su valor es verdadero se añaden checkboxes. Esta propiedad, como la anterior, sirve para las relaciones. Un buen uso sería para enumeraciones o colecciones de datos que se sepa de antemano que no van a contener una gran cantidad de instancias indicar el atributo `short`, si el número de instancias que almacena la entidad de la relación es elevado es recomendable usar el atributo `dropdown`.
- El atributo `file`, indica que se debe añadir un campo para la subida de ficheros en la GUI. A la hora de usarla, su valor debe ser `single` (si solo se quiere subir un fichero) o `multiple` (si se quieren subir varios documentos).

- Junto con el atributo anterior se puede usar `image`, este atributo indica si se pueden subir toda clase de ficheros o solo imágenes. Para usarla, su valor debe ser verdadero para permitir solo imágenes. Si se quieren subir todo tipo de ficheros, no se indica.
- El atributo `map` indica la necesidad de añadir un mapa . Cuando se añade, en los formularios de editar y crear y en la pantalla de detalles se muestran dos pestañas. En la segunda pestaña hay incrustado un mapa de Google Maps al que se le pueden añadir o eliminar marcas y ver un pequeño listado con todas ellas.

```
<table name="Contacto" pluralName="contactos" icon="fa fa-envelope" sqlTableName="contactos"
       label="Contacto" pluralLabel="Contactos" syncMode="none" generateSymfonyModule="true"
       genre="male">
    <column name="id" label="ID" sqlColumnName="id" type="PKINT" primaryKey="true"
       autoIncrement="true"/>
    <column name="nombre" type="VARCHAR" size="255" label="Nombre" listable="true"
       searchable="true" form="true"/>
    <column name="email" email="true" type="VARCHAR" size="255" label="Email" listable="true"
       searchable="true" form="true"/>
    <column name="telefono" type="PHONE" label="Teléfono" listable="true" searchable="true"
       form="true"/>
    <column name="supervisor" dropdown="true" type="PKINT" label="Supervisor" description="true"
       listable="true" searchable="true" form="true"/>
</table>
```

Figura 4.5: Ejemplo de una tabla con un subconjunto de los atributos descritos.

Algunos de los atributos personalizables, solo son usados en el código por lo que no son mostradas en la GUI de la web, los otros son solo para poder mostrarlos en la web. Hay atributos cuyo uso es obligatorio y si no se indican se muestra un error, sin embargo, hay otras que son opcionales y si no se indican simplemente no modifican el contenido de la página web. En la figura 4.5 se muestra un ejemplo de uso de algunos atributos.

En el fichero XML, también se deben añadir las relaciones entre las entidades, para que se puedan añadir clases Angular2 intermedias en la aplicación web si es necesario. En las relaciones se ha añadido un atributo:

- El atributo `pkRelationName`, permite indicar el nombre que se le quieres dar a la relación en un sentido. Esto se usa en la generación de la interfaz gráfica, a la hora de mostrar la relación en un selector (o dropdown) en el formulario de creación y editado (para indicar el nombre del input) y en el código. Un ejemplo sobre cómo usar esta etiqueta se encuentra en la figura 4.6

```
<relation name="Contacto_Supervisor" multiplicity="ONE_TO_MANY">
    <pkTable>Contacto</pkTable>
    <pkColumn>id</pkColumn>
    <pkRelationName>subordinados</pkRelationName>
    <foreignTable>Contacto</foreignTable>
    <foreignColumn>supervisor</foreignColumn>
    <foreignRelationName>supervisor</foreignRelationName>
</relation>
```

Figura 4.6: Ejemplo de la creación de una relación.

Internamente se ha hecho una conversión de tipos. En el fichero XML se le han indicado los tipos que se usan en la base de datos, pero Angular2 no es compatible con estos tipos y hay que indicarlos para que los interprete de forma correcta. Para las fechas no se ha creado ninguna etiqueta específica, con solo saber su tipo de dato, se crea directamente un selector de fechas ("date picker") en los formularios y se muestra la fecha formateada en los listados.

### 4.2.2. Estructura del generador

En la figura 4.7 se muestra el diagrama de actividades de la aplicación a la hora de leer los datos que el programa necesita y generar el administrador.

Para generar automáticamente toda la estructura de ficheros del administrador, se ha usado un conjunto de ficheros creados bajo el misma carpeta denominada "generador". En este directorio los ficheros se han agrupados según la funcionalidad que generan en el administrador.

Se ha creado un directorio donde se han almacenado las plantillas que generan lo propio de una sola entidad. En esta carpeta se han incluido las plantillas para la generación de la clase de la entidad con todos sus atributos, las funcionalidades específicas de editar y crear (para ambas funciones se usa la misma plantilla), listar y detalles (para estas plantillas se ha creado tanto los componentes con la funcionalidad como el modelo con la vista), el servicio que comunica el administrador con la base de datos, las rutas usadas (la navegación entre las páginas de la entidad) y el módulo. Por cada entidad se generaran todos estos ficheros.

Se ha creado otra carpeta que contiene la plantilla con el enrutado global (el padre) para toda la aplicación, en ella se enlazan las rutas de cada una de las entidades, otra plantilla que genera el menú lateral general de la aplicación y el fichero de estilos.

Como los ficheros están disgregados en diferentes carpetas, para una ejecución rápida del generador, se ha creado un fichero PHP encargado de lanzar todos los ficheros. De esta forma, con una única instrucción ejecutada en la ruta donde se encuentra el lanzador de la aplicación se pueden ejecutar todos ellos: `php main-generator.php`.

### 4.2.3. Estructura de los administradores

En esta subsección se muestra a grandes rasgos el diagrama de actividades del administrador generado, figura 4.8. Se ha tratado de abstraer al máximo y hacerlo lo más general posible.

Al seleccionar una entidad aparece es un listado con todas sus instancias guardadas en la base de datos. A partir de esta página se puede crear una instancia, para ello aparece un formulario que hay que rellenar. En caso de que la entidad seleccionada tenga mapas, se puede añadir marcas clicando en el mapa, todas las marcas creadas aparecen relacionadas en la misma página. Se pueden eliminar las coordenadas, clicando en la opción de eliminar. Si la entidad contiene traducciones, también se le pueden añadir indicando el nombre de la traducción y el idioma al que pertenece (español, francés, alemán o inglés). En caso de querer eliminarla, seleccionar la opción de eliminar.

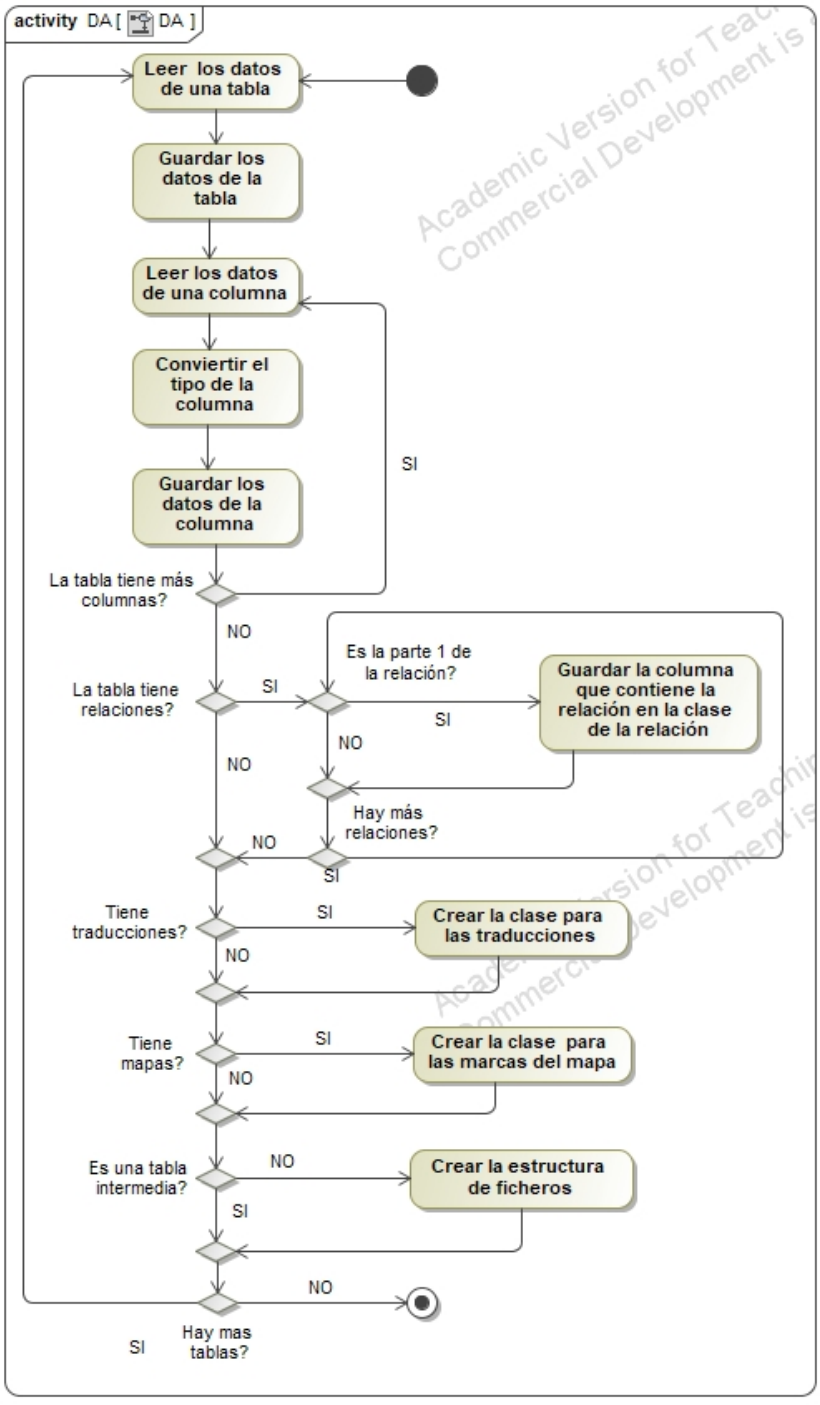


Figura 4.7: Diagrama de actividades del generador.

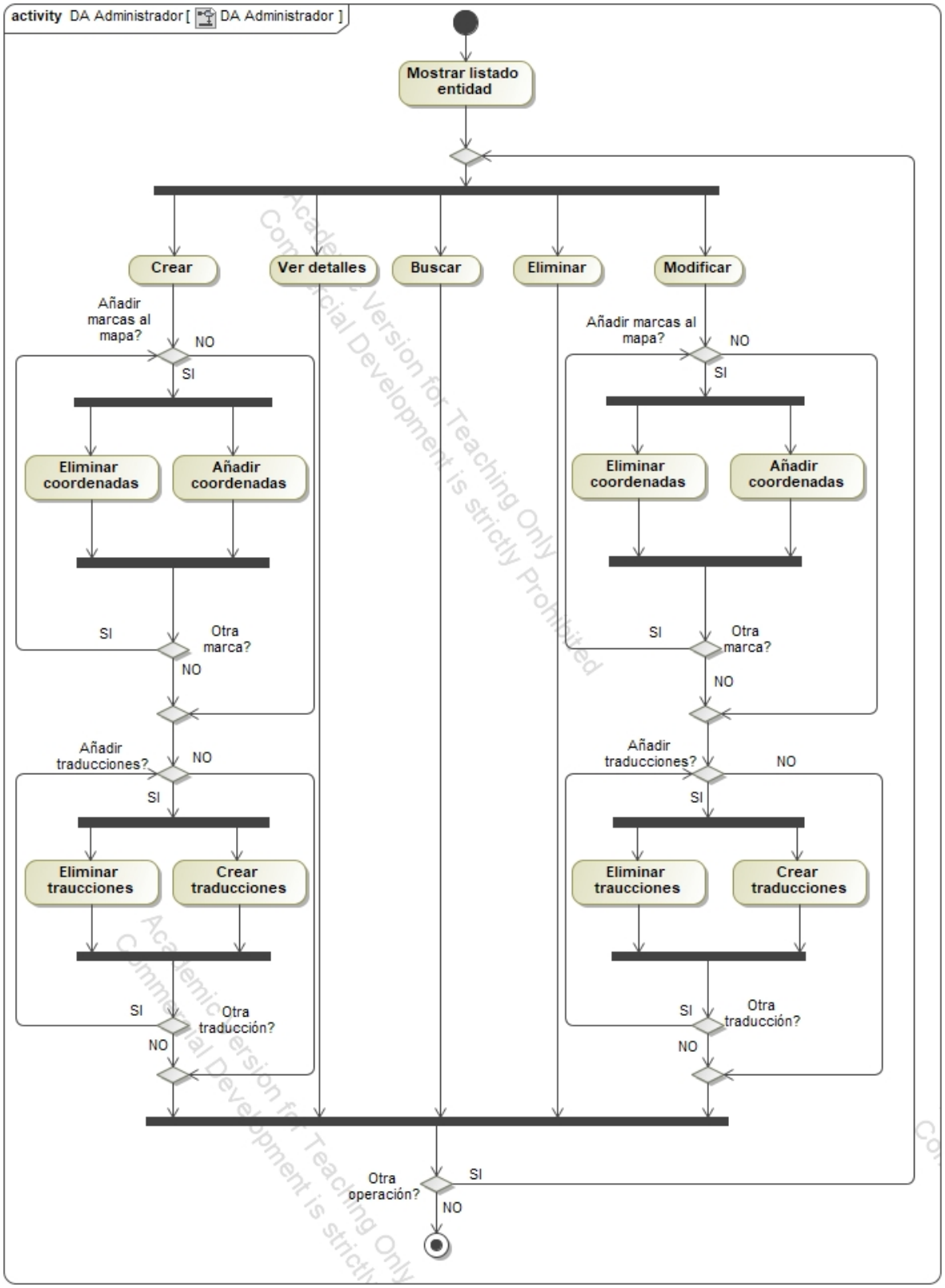


Figura 4.8: Diagrama de actividades del administrador.

Si lo que se quiere es modificar una instancia, se tendrá que seleccionar la opción de editar la instancia deseada. El proceso para la edición de las instancias es el mismo que el anterior, pero en vez de partir de formularios vacios dichos formularios estarán rellenados con los datos de la instancia.

En la página del listado también están las opciones de ver los detalles de una instancia, lo que nos lleva a una página con el listado de todos sus atributos (no se pueden modificar en este caso), de eliminar una instancia y de buscar.

### 4.2.4. Estructura de la base de datos

En una aplicación como esta, cuya utilidad principal radica en la generación automática de administradores, es el usuario quien aporta como dato de entrada el fichero XML con la estructura de la base de datos. Por ello, no hay un diagrama de clases específico, sin embargo para probar que la aplicación funciona correctamente se ha hecho uso de uno. En él se ha tratado de introducir todos los tipos de datos diferentes que existen así como relaciones "uno a uno", "uno a muchos" y "muchos a muchos".

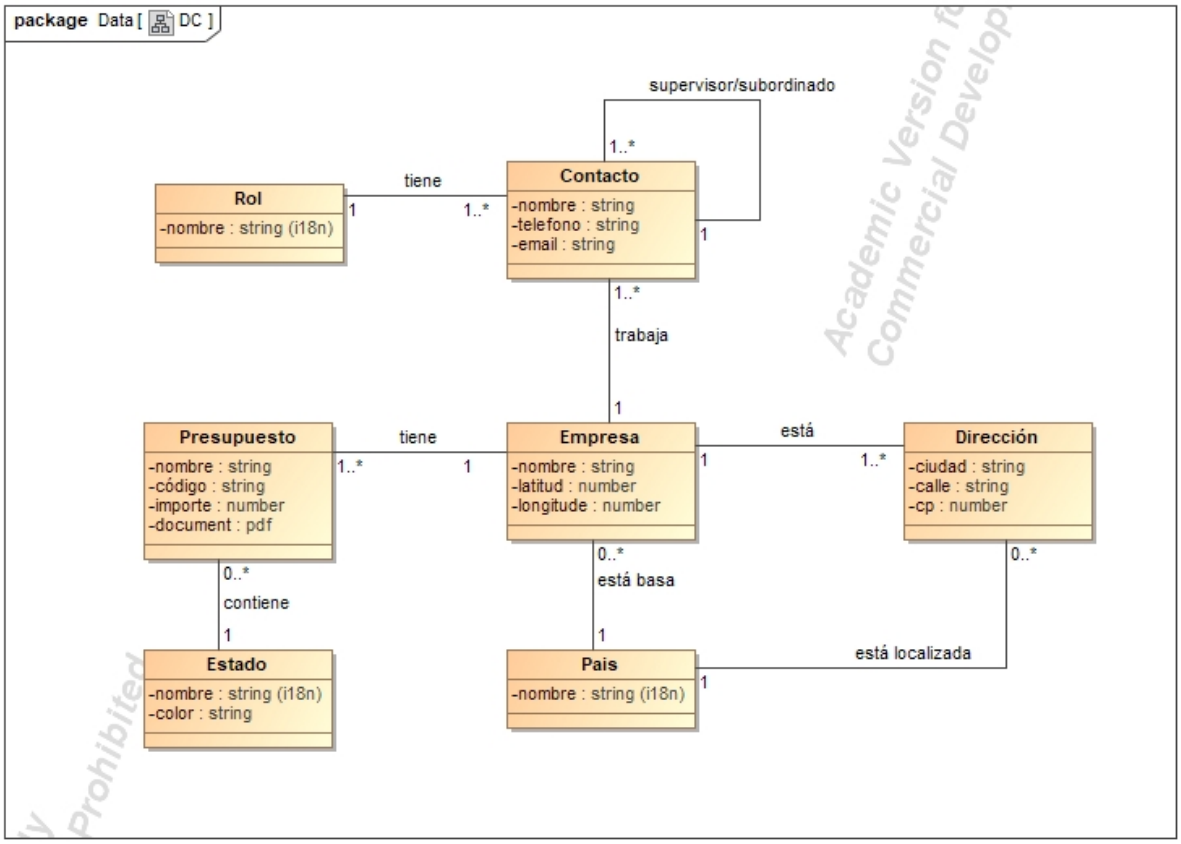


Figura 4.9: Diagrama de clases de un administrador de empresas.

En este ejemplo concreto se pretende crear un administrador para gestionar una empresa formada por contactos (o trabajadores) que pueden ser supervisores o subordinados y tienen un rol dentro de la empresa. Una empresa puede estar localizada en diferentes direcciones o

incluso en diferentes países. Para cada empresa se guardará en qué país está localizada la sede principal. Una empresa tiene diferentes presupuestos con sus gastos y su estado. El estado de un presupuesto indica si se ha excedido o si está dentro de los límites. Un presupuesto en un momento dado solo puede tener un estado.

Como se puede ver en el diagrama de la figura 4.9, se ha intentado persistir la mayor cantidad diferente de datos, así como de relaciones. Hay relaciones de "uno a muchos", de "muchos a muchos", de "uno a uno" e incluso una relación recíproca. En cuanto a los datos hay texto, traducciones (i18n), colores, imágenes, documentos y fechas.

## 4.3. Diseño de la interfaz

En este apartado se va a explicar el diseño de la interfaz de usuario de la aplicación desarrollada. Como en el subapartado anterior, se va a mostrar un ejemplo concreto. De hecho se muestra la interfaz gráfica del mismo administrador.

Al abrir la aplicación aparece la página mostrada en la figura 4.10, en la parte de la izquierda hay una barra lateral (left sidebar), esta barra se muestra y oculta automáticamente dependiendo del tamaño de pantalla. El usuario también puede interactuar con ella clicando en el botón de las tres rayas o pasando el ratón por encima. En la parte superior aparece el nombre que el usuario ha querido dar al administrador. Cuando la barra lateral está desplegada se muestra el nombre de todas las entidades indicadas en el XML junto con sus iconos, cuando está oculta únicamente se muestran los iconos.

En la parte central de la pantalla aparece el listado de la primera entidad declarada en el XML. En la parte superior de cada columna de la tabla se puede observar un campo para la búsqueda de instancias por sus atributos. En la columna de la derecha están los botones que permiten la edición, borrado y visualización de cada una de las instancias, en la parte inferior de la tabla está la paginación. El usuario puede establecer el número de instancias que se deben mostrar (10, 15, 20, 30 o 50), esto ha sido pensado para evitar el desplazamiento hacia abajo y arriba de la página (scroll). Arriba del nombre de la tabla aparece un breadcrumb (miga de pan), esto indica al usuario en qué zona de la página web se encuentra y cómo ha llegado a ella. Y debajo de la tabla está el botón para crear nuevas entidades.

Al crear una nueva entidad aparece un formulario como el que se muestra en la figura 4.11. Este formulario cambiará dependiendo de los elementos que el usuario haya indicado. En la parte superior, como en la pantalla anterior, aparece el breadcrumb mediante el que podemos navegar hacia atrás en la aplicación. Al intentar borrar una instancia aparece una ventana modal para informar al usuario de los cambios que se van a producir. La figura 5.6 muestra los detalles del contacto.

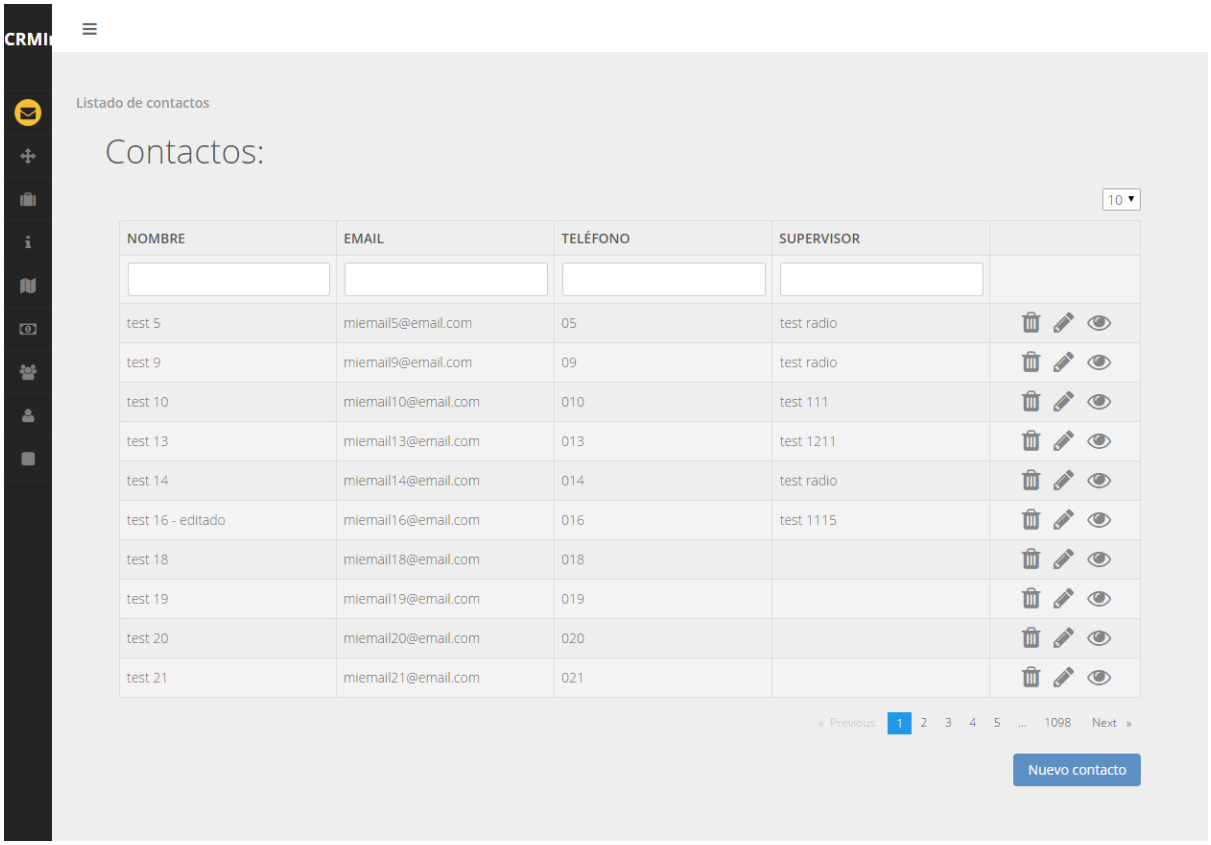


Figura 4.10: GUI página de listado de la entidad contacto.

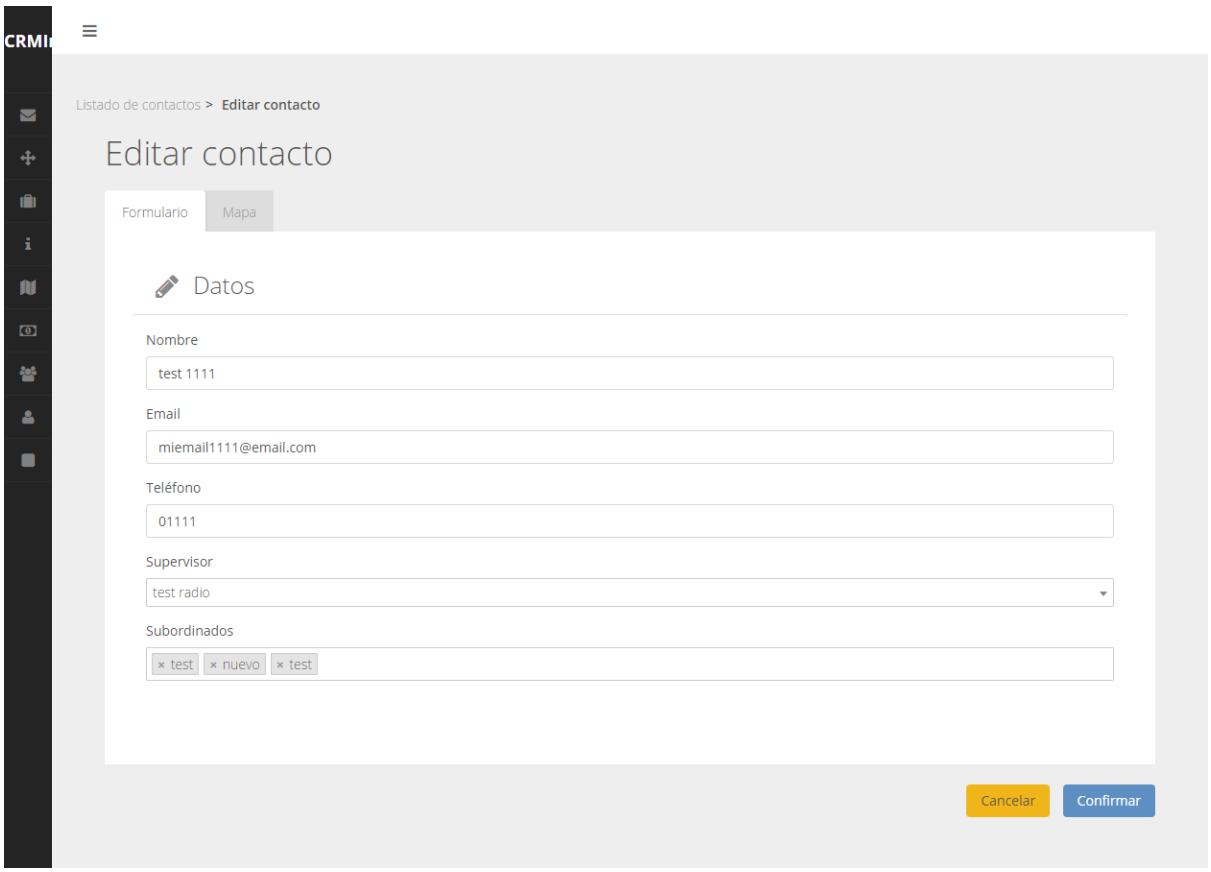


Figura 4.11: GUI página de editado de la entidad contacto.

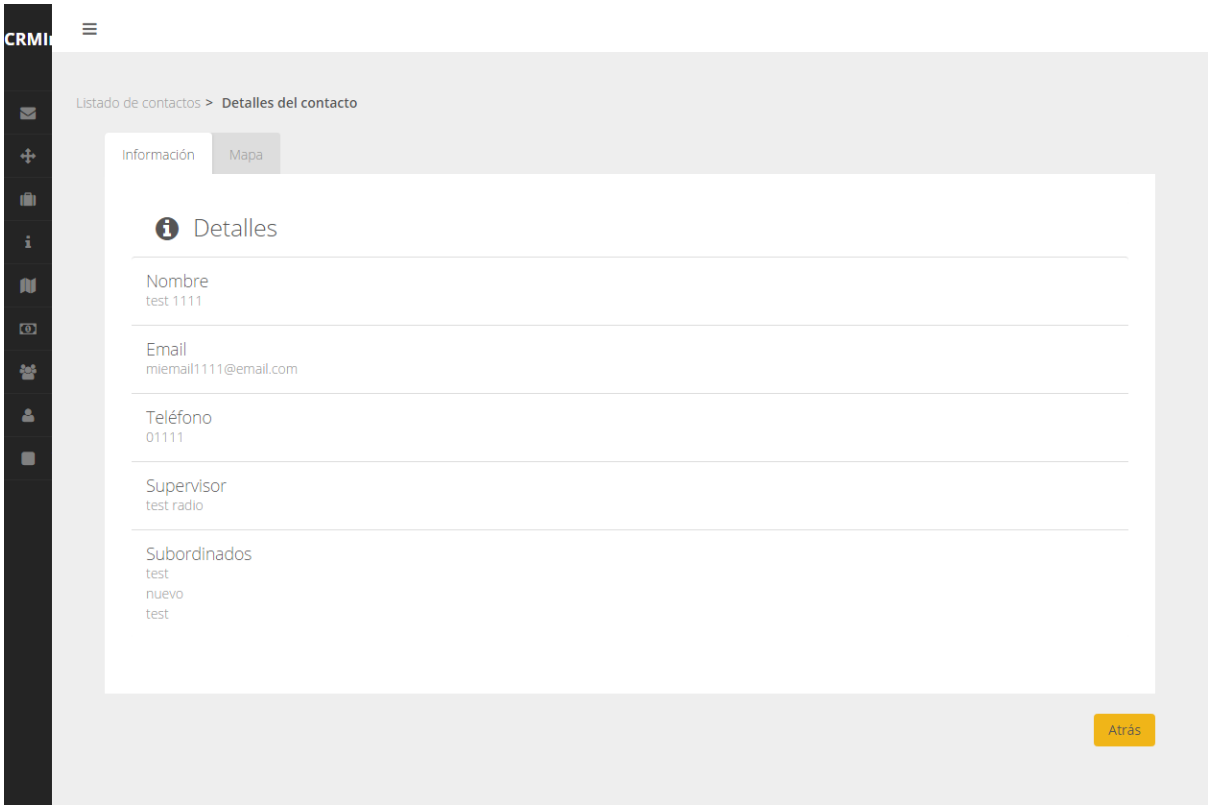


Figura 4.12: GUI página de detalles de la entidad contacto.

# Capítulo 5

# Implementación y pruebas

## 5.1. Entorno de ejecución

Durante este proyecto se ha desarrollado primero un administrador concreto de forma manual y posteriormente el generador de administradores (que era el objetivo). El primero se ha desarrollado únicamente a modo de formación, para entender que es un administrador y que es lo que se espera de él.

Para cada uno de ellos se ha creado un entorno de trabajo en el que poder lanzar la aplicación, ver su funcionamiento y ejecutar las pruebas. Ambos entornos usan un conjunto de herramientas muy parecido. Se ha trabajado en el editor de texto WebStorm que incorpora algunas de las herramientas que se van a usar como plugins para el autocompletado y corrección de errores de Angular2, HTML, TypeScript y CSS, y una terminal muy usada a la hora de importar los componentes externos, además para el generador se ha añadido el plugin de Twig. Se ha incorporado nodeJS y Angular CLI que facilitan en gran medida la gestión del proyecto. Finalmente para la parte de PHP, se ha instalado las herramientas WAMPP y Composer.

## 5.2. Desarrollo de los sprints

En esta sección se muestra como se ha implementado la aplicación y los contratiempos que han surgido. Se detallan todos los sprints que se han llevado a cabo durante el desarrollo de la aplicación web. En los sprints, se muestran todas las tareas planeadas para llevar a cabo y la historia de usuario relacionada. La duración de los sprints dependía de la disponibilidad del responsable del proyecto, a veces se adelantaban o retrasaban por lo que no siempre duraban una semana.

### 5.2.1. Sprint 1

Para el primer sprint (ver cuadro 5.1) se decidió empezar leyendo el XML con PHP, ya que era lo más básico. Para ello se empezó investigando y mirando PHP junto con SimpleXML y se crearon las etiquetas que se consideraron para poder leer la información de la forma más sencilla. En este mismo sprint se decidió escoger una plantilla HTML y añadirla al nuevo proyecto.

| Tarea | Descripción |
|---|---|
| 1 | Leer el XML (nombre de entidades y atributos de cada entidad junto con sus características). - HU_GEN_01 |
| 2 | Añadir la plantilla. - HU_ADM_08 |
| 3 | Crear las etiquetas necesarias para el fichero XML. - HU_GEN_01 |

Cuadro 5.1: Sprint 1

### 5.2.2. Sprint 2

Como se terminaron todas las tareas del sprint anterior a tiempo, no se arrastró ninguna. Sin embargo, el responsable del proyecto avisó de que seguramente la tarea tres se debería reabrir más adelante por falta de etiquetas que de momento no eran necesarias. En el segundo sprint (ver cuadro 5.2) se empezó creado la estructura base del proyecto con plantillas Twig.

| Tarea | Descripción |
|---|---|
| 4 | Routing (enrutado de la aplicación). - HU_ADM_07 |
| 5 | Module (app.module.ts con todos los componentes, servicios, módulos externos). - HU_ADM_01, HU_ADM_02, HU_ADM_06 |
| 6 | Entidades junto con sus atributos. - HU_GEN_02 |

Cuadro 5.2: Sprint 2

### 5.2.3. Sprint 3

La tarea 5 del sprint anterior se realizó con éxito gracias al administrador manual parcialmente realizado anteriormente, ya que contenía código por defecto. Además ya se tenían claros qué módulos externos que se iban a usar y la organización de todas las páginas dentro de la aplicación. A la hora de la validación se aconsejó separar el routing padre en varios hijos, uno para cada entidad y así tener el código agrupado. Por lo que la tarea 4 se arrastró para el siguiente sprint.

En el tercer sprint (ver cuadro 5.3) , se empezó a desarrollar las operaciones de crear y editar las entidades y la parte de la comunicación con la base de datos relacionada con estas operaciones. En este punto se decidió reabrir las tareas 3 y 1 porque se necesitaban más datos que no se habían tenido en cuenta. Además se llevaba del sprint anterior la tarea 4 pendiente.

| Tarea | Descripción |
|---|---|
| 1 | Leer el XML (leer las características no contempladas de las entidades: plural, género). - HU_GEN_01 |
| 3 | Crear las etiquetas necesarias al XML (plural, singular y genero de la entidad). - HU_GEN_01 |
| 4 | Routing (el padre y los hijos). - HU_ADM_07 |
| 7 | Editar y crear las entidades: Formularios, pestañas, selectores múltiples. - HU_ADM_02, HU_ADM_06 |
| 8 | Servicios (comunicación cliente-servidor): obtener las entidades por su identificador, guardar la entidad en la base de datos y añadir una nueva entidad. - HU_ADM_02, HU_ADM_06 |

Cuadro 5.3: Sprint 3

### 5.2.4. Sprint 4

A la hora de la validación del sprint anterior, el responsable comentó la utilidad de añadir mapas y traducciones a las entidades que se quisieran. Por lo que la tarea 7 arrastró para el nuevo sprint (ver cuadro 5.4). Las tareas 1 y 3 también tuvieron que ser arrastradas ya que se necesitaba crear nuevas etiquetas y leer nuevos valores, para saber si la entidad contenía traducciones o mapas. La tarea 8 se arrastró ya que en el sprint anterior solo se habían realizado las peticiones de creación y editado de las entidades.

Únicamente se pudo escoger la tarea nueva de listar entidades para que no se acumulara el trabajo y poder terminarlo todo.

| Tarea | Descripción |
|---|---|
| 1 | Leer el XML (leer las características no contempladas de las entidades: mapas y traducciones). - HU_GEN_01 |
| 3 | Crear las etiquetas necesarias al XML, una para los mapas y otra para las traducciones. - HU_GEN_01 |
| 7 | Editar y crear las instancias: añadir mapas. - HU_GEN_02, HU_GEN_06 |
| 8 | Servicios (comunicación cliente-servidor): listar entre límites y búsquedas. - HU_ADM_02, HU_ADM_06 |
| 9 | Listar las entidades: Añadir una tabla (datatable), paginación, búsquedas por todo, filtros por columnas, mapas y pestañas - HU_ADM_01 |

Cuadro 5.4: Sprint 4

### 5.2.5. Sprint 5

Todas las tareas del sprint anterior fueron validadas de forma satisfactoria y se escogieron todas las tareas que quedaban. Se pensó que este sprint iba a ser más corto y se podría dedicar el tiempo restante al diseño de la web, pero la tarea 11 fue mucho más compleja de lo que se esperaba.

| Tarea | Descripción |
|---|---|
| 10 | Detalles de las entidades: Añadir mapas, pestañas y listas estructuradas. - HU_ADM_03 |
| 11 | Implementar las relaciones entre las entidades. - HU_ADM_06, HU_ADM_03, HU_ADM_02 |

Cuadro 5.5: Sprint 5

## 5.3. Detalles de implementación

El administrador está formado por tantas entidades como el usuario indique. Estas entidades (o clases en Angular2) se agrupan en una única carpeta. A partir de ellas se generan el resto de clases necesarias que siempre son las mismas: el modelo (o template) y el fichero que da la funcionalidad para cada una de las funciones de editar y crear, listar y los detalles de la entidad.

Para que los componentes no dependan unos de otros y hacer inyecciones de dependencias en todos ellos, se crea un servicio para cada uno. El servicio, además de mantener los componentes sin relaciones entre ellos, permite encapsular todas las llamadas a la base de datos. De esta manera se mantienen todas las peticiones organizadas en el mismo fichero.

Se crea otro fichero en cada entidad para el routing. En un principio se pensó establecer un routing global que sirviera para toda la aplicación. Pero, a medida que el esquema iba incrementando, se prefirió separarlo en un padre situado en la raíz del proyecto y un hijo por cada entidad declarada en el fichero XML.

En cada componente, se crea un fichero llamado módulo (del inglés module.ts). Este fichero importa todos los módulos externos y los componentes internos usados así como los servicios inyectados. Este fichero, como el anterior, se decidió que era mejor separarlo para así importar solo los módulos externos que se vayan a usar en los componentes en los que el usuario lo haya indicado.

Para la conexión con la base de datos, se ha hecho el diseño de la API Rest como el que se muestra en la figura 5.6. Se ha hecho que todas las url sean cortas y claras para el usuario. En los servicios se ha declarado una constante que contiene la parte fija de la url, al realizarse las peticiones se concatena esta parte fija con la parte variable propia de cada método. Como la API Rest se genera automáticamente solo se muestra lo necesario para una entidad genérica.

A la hora de generarla se ha tratado de seguir el estándar para el diseño de una API Rest [5]. Este estándar indica que las operaciones POST sirven para la creación de los recursos, las GET se usan para la recuperación de los datos, las PUT para la edición y las DELETE para eliminar.

En este caso no se ha hecho uso de operaciones PUT, la recuperación de un recurso se ha hecho mediante una operación GET como la creación. Otra variación se encuentra en que al buscar una instancia de la base de datos se hace mediante una petición POST. No se hace mediante el método GET porque se pueden buscar las entidades por todos los atributos indicados como “serchables”, por lo que se tendrían que pasar todos ellos en la url (se haya buscado por ellos o no) y se deberían pasar en el mismo orden en que se han especificado en el recurso (cosa

| Método | URL | Excepciones | Descripción |
|---|---|---|---|
| GET | /nombre_entidad/{offset}/limit/list | 200 | Se devuelven todas las entidades de la base de datos |
| GET | /nombre_entidad/{id}/show | 200 y 404 | Se devuelve la instancia cuyo id se corresponda con el del recurso |
| POST | /nombre_entidad/{locale}/{id}/update | 200 y 404 | Se actualizan los datos de una instancia |
| DELETE | /nombre_entidad/{locale}/{id}/delete | 200 y 404 | Se elimina la instancia cuyo id se corresponda con el del recurso |
| POST | /nombre_entidad/{locale}/new | 200 | Se crea una nueva instancia |
| POST | /nombre_entidad/search | 200 y 404 | Se devuelve la entidad buscada |

Cuadro 5.6: API RESTfull de la base de datos.

que podría llevar a errores difíciles de detectar). A la hora de realizar la petición POST se pasa un JSON como cuerpo de la petición con todos los parámetros y si solo se ha hecho la búsqueda por un subconjunto de los atributos los demás se dejan vacíos.

Se ha hecho uso de un conjunto muy limitado y básico de excepciones para informar al usuario de la respuesta del servidor frente a una petición. La respuesta con código 200 indica que todo ha ido bien, por lo que la base de datos ha hecho de forma satisfactoria la petición demandada y devuelto o eliminado el objeto. La respuesta de error con código 404 indica un recurso no encontrado, se puede producir al buscar una entidad por su identificador debido a que no se encuentra en la base de datos (porque se ha borrado o el identificador es incorrecto).

En las peticiones, se ha supuesto que la instancia tiene asignadas las traducciones, en caso de no tenerlas la API se generaría sin este parámetro. El parámetro denominado "nombre_entidad" se corresponde con el nombre de las entidades especificadas en el fichero XML. Este conjunto de rutas se genera para cada entidad.

## 5.4. Módulos externos

El responsable del proyecto pidió una serie de características para el administrador generado que requerían una gran cantidad de horas para implementarlas (como paginación o mapas), así que se decidió añadir módulos externos al proyecto.

Para añadir un módulo primero se deben investigar todos los que hay disponibles en los repositorios (en este proyecto todos los módulos usados se han obtenido del repositorio GitHub) y elegir el que mejor se adapta a la versión de Angular2 que se está usando. También se debe mirar que las incidencias del proyecto se vayan resolviendo y que las fechas de las modificaciones sean recientes, para asegurarse de que haya al menos una persona trabajando en él. La documentación

es esencial, que no solo esté indicado cómo empezar a usar el módulo (el apartado de "get started") sino que tenga ejemplos prácticos y la API documentada. En algunas ocasiones, se perdía más tiempo investigando el módulo que incorporándolo al proyecto.

Todos los módulos, al descargarse, se añaden dentro de la carpeta "node_modules" y además se deben incorporar al fichero "app.module.js" manualmente. Para ejecutar el generador de administradores por primera vez, al descargarlo del repositorio se debe ejecutar la instrucción `npm install` para que descarguen todos los módulos externos en la versión que aparece en el "package.json". Para cambiar la versión un módulo externo, o bien se puede modificar este fichero indicándole la versión que se quiere y volver a ejecutar la instrucción `npm install`, o bien mirar en la documentación del módulo la instrucción necesaria para ello. Para eliminar un módulo, se puede borrar la línea del fichero de módulos y del fichero de paquetes y borrar la carpeta propia del módulo (para que no ocupe espacio innecesario).

En los siguientes subapartados se explican todos los módulos externos usados en la aplicación.

### 5.4.1. ng2-pagination

En un principio se propuso el uso de una datatable que podría hacer las cosas mucho más sencillas. El datatable es un elemento que viene con todo lo necesario para listar todo tipo de objetos complejos. Le añade la paginación a la tabla, el que se usó también le añadía campos de búsqueda arriba de cada una de sus columnas. Suelen tener otro buscador general fuera de la tabla (generalmente en la parte superior derecha de la tabla) y el usuario tiene la posibilidad de ordenar todas las columnas de mayor a menor y al revés. Todo esto incluido con el estilo de bootstrap3 y únicamente se le tiene que pasar un conjunto de datos.

La dificultad del datatable que se seleccionó era que estaba situado en el lado del cliente, por lo que cuando se cargaba la tabla por primera vez tardaba bastante tiempo en leer todos los datos. Como era un módulo externo, no se tenía control sobre lo que realizaba internamente, únicamente se podía acceder a su API y ejecutar los métodos establecidos. Por ello se decidió prescindir del módulo e incluir una tabla de Bootstrap normal a la que se le añadieron los campos para las búsquedas por atributos y un módulo externo para la paginación.

El módulo externo "ng2-pagination" permite paginación en el lado del servidor, por lo que únicamente cargará y mantendrá en memoria una página de la tabla. De esta forma aunque se hagan más peticiones al servidor, los datos se actualizan cada vez que pasas de página. Al hacer este cambio se modificó la API Rest para que en el listado se pudieran pasar los parámetros que indican el índice de la primera instancia y el número de instancias a cargar en la tabla (offset y limit).

### 5.4.2. angular2-google-maps

Este módulo permite incrustar en cualquier página de la aplicación un mapa de Google y contiene un conjunto de métodos con los que poder personalizar el mapa. Lo primero que hay que hacer para poder empezar a usar este módulo es pedir a Google una clave "api key", para

ello únicamente hay que indicar el nombre de un proyecto y se muestra la clave.

Existen diferentes claves, en este caso se escogió una clave gratuita (ya que no se espera un gran número de peticiones) para una aplicación web con JavaScript. También hay disponible una clave para móviles (ya sean Android o iOS). Esta clave se necesita indicar al importar el módulo y con ello ya se puede incrustar el mapa en la interfaz.

Ya se ha explicado en secciones anteriores que para indicar la existencia de un mapa en una entidad se debe añadir la etiqueta "map" junto a una columna. Al principio se pensó usar esta etiqueta junto a otras etiquetas que te permitieran elegir el color y una imagen, modificando el icono del pin por defecto. Para cambiar el icono y sustituirlo por una imagen en el mismo módulo hay una función que te permitía hacerlo.

Para cambiar el color se planteó hacer uso de peticiones al servidor de Google, un ejemplo de uso es la siguiente url `http://chart.apis.google.com/chart?chst=d_map_pin_letter&chld=a|0000FF`. Esta petición establece el pin por defecto en color azul y en vez del punto negro aparece la letra "a" tal y como se ha especificado. Con ello se pueden crear marcas personalizadas en caliente a medida que se van leyendo los datos de la base de datos.

También se pensó que en vez de tener diferentes localizaciones (marcas en el mapa) por cada instancia de la entidad permitir solo una, de manera que el usuario pudiera decidir si quería ver las entidades listadas en una datatable o distribuidas por el mapa y al clicar en un pin se detallara su información.

### 5.4.3. angular2-select

Este módulo externo es un selector parecido al que viene por defecto en HTML pero en vez de mantener la lista de elementos a seleccionar en local, realiza una serie de peticiones a la base de datos cada cierto tiempo si detecta que el texto introducido se ha modificado respecto a la última petición. Esto puede llegar a saturar la base de datos con peticiones si constantemente se está modificando el texto, pero es mejor que tener que descargarse toda una tabla para poder seleccionar un solo elemento.

Este elemento se ha usado para las relaciones, de "uno a uno" y de "uno a muchos", cuando se debe elegir un elemento de otra tabla. En algunos casos está delimitado a poder elegir únicamente un elemento, en otros se deja elegir todos los elementos que se quiera sin un límite establecido.

### 5.4.4. ng2-file-upload

Este módulo permite la subida de ficheros a un servidor. Primero se suben los ficheros al servidor donde se almacenan. Esta operación devuelve una url por documento indicando dónde están almacenados y el nombre que se les ha dado. Cuando se guarda el formulario donde se ha incluido este módulo, en el campo de los documentos de la base de datos se le añade la url devuelta. El hecho de almacenar los ficheros aparte, hace la gestión de los ficheros mucho más

simple e independiente de la base de datos.

Este módulo viene preparado para poder elegir el tipo de fichero que se quiere subir (imágenes o documentos de texto), también se puede delimitar a subir solo un fichero o un conjunto de ficheros. Si se permite subir varios ficheros, los sube todos en la misma petición en vez de subir uno por uno. Si solo se quiere añadir dos o tres ficheros no pasa nada, pero si se quiere añadir un número excesivo de ficheros el servidor se saturaría con todas las peticiones.

### 5.4.5. ng2-breadcrumb

Este es un módulo que añade en la página una cadena de texto con la que puedes navegar, esta cadena de texto de va creando a medida que vayas entrando en la web. En un administrador hay pocos niveles de profundidad por lo que no es imprescindible su uso, sin embargo se ha decidido usar por temas de navegación dentro de la aplicación, además indica al usuario donde se encuentra en todo momento. Es una manera eficaz de decirle que es lo que está haciendo en cada momento.

## 5.5. Validación y verificación

Para validar que el generador funcionaba de forma correcta a medida que se iba creando, se creó un fichero XML con todas las etiquetas diseñadas para ver cómo se alteraban los formularios. Al final, se creó un nuevo fichero XML diferente para ver como generaba automáticamente el administrador.

Por otra parte se había hecho previamente un administrador de forma manual en el que se había probado todas sus funcionalidad para comprobar que funcionaba correctamente. Por lo que se comprobó que el código del administrador generado fuera el mismo o prácticamente el mismo solo que más refactorizado.

Además de probar las funcionalidades de los administradores, se hizo una prueba de estrés. Para ello se escogió un tipo de entidad a la que se le añadieron mil objetos en la base de datos, y se probaron de nuevo todas las operaciones para esta entidad comprobando que los tiempos no fueran excesivos. Con esta prueba se testeaba la eficiencia de las operaciones realizadas y como manejaba una gran cantidad de datos. Preocupaba sobretodo la respuesta del selector por sus constantes peticiones y porque la consulta debería ser casi instantánea.

Para la verificación, la metodología ágil basada en Scrum que se usó a la hora de desarrollar la aplicación facilitó mucho la tarea, ya que semanalmente se enseñaba la aplicación al supervisor del proyecto para que comprobara que la GUI se adaptaba a lo esperado

# Capítulo 6

# Conclusiones

Con este proyecto, me he dado cuenta lo difícil que es planificar, estimar la duración de todas las tareas, sobre todo cuando no tienes experiencia en el lenguaje (o lenguajes) de programación usados (en este caso los frameworks Angular2 y Twig y el lenguaje de programación PHP). Se necesita saber si será fácil la ejecución de una tarea o incluso si se puede hacer. En algunas ocasiones te das cuenta que se puede hacer de diferentes formas y debes saber para cada caso particular, cuál es la forma más sencilla y eficiente. Prácticamente tienes que saber cómo vas a desarrollar el código desde el principio, para evitar contratiempos. Por ello se decidió usar una metodología ágil, donde se permite una planificación flexible que se adapte a los contratiempos con los que te puedes encontrar.

La definición de requisitos y el diseño del sistema son indispensables para una buena planificación. Se debe establecer qué es lo que se espera del sistema a desarrollar, esta parte es la base del proyecto, los fundamentos sobre los que se desarrollará. No nos podemos permitir dejar sin aclarar algún aspecto que no se haya entendido perfectamente, porque esto sólo producirá cambios en la planificación y retrasos. Debemos tener claro que lo que nosotros pensamos que será el futuro sistema, es lo mismo que lo que el cliente quiere obtener. Y si hay alguna cosa que creamos que se puede mejorar con respecto a las peticiones del cliente, comentárselas y explicarselas para que las acepte o rechace.

Me ha gustado el dinamismo de Angular2, cómo se pueden hacer páginas muy dinámicas sin tener que escribir nada de jQuery (o JavaScript) únicamente con las funciones que te proporciona Angular2. De Twig me ha sorprendido su flexibilidad. Al principio no entendía el concepto de las plantillas y de generar código de forma automática, de tener ficheros con la extensión propia de Twig que se pudieran ejecutar dando lugar a ficheros en otros lenguajes de programación.

Durante la estancia en prácticas en la empresa Cuatroochenta, me ha sorprendido el compañerismo y lo bien que se llevan todos los trabajadores. Pensaba que al ser una empresa privada, habría mucha competencia entre ellos para intentar demostrar que son más capaces que el resto. Pero a medida que me fui integrando en el equipo, me di cuenta que si tenía cualquier problema todos los compañeros trataban de ayudar empleando su tiempo libre o de trabajo. Por ejemplo a la hora de añadirle mejoras a la aplicación web desarrollada o a la hora de imputar horas y crear tareas en Jira.

## 6.1. Futuro del proyecto

La principal y más urgente mejora es la automatización de la generación de la base de datos que no dio tiempo durante la estancia en prácticas, únicamente se proporcionaron un conjunto de tablas con todas las variaciones de datos reconocidas en el XML.

También creo que es necesario añadir una página principal, algo así como un "dashboard" e incluir estadísticas muy generales. Un conjunto de etiquetas (una por cada tabla de la base de datos) con el icono que se le haya dado a la tabla, su nombre y el número de entidades totales que hay o las creadas durante el mes anterior. Simplemente para ver el uso del administrador.

Para que este administrador pueda ser usado por todo tipo de público, no solo aquellos con conocimientos informáticos, creo que sería muy interesante añadir una aplicación que permita generar el fichero XML mediante una interfaz gráfica. Formularios dinámicos que permitan indicar los nombres de las tablas, sus columnas y elegir todos los tipos de datos que se quieren almacenar.

Como algo complementario, es interesante la subida de vídeos y la realización de una página de galería que permita visualizar (sin necesidad de descargar) todas las imágenes, ficheros y vídeos subidos.

# Bibliografía


[1] Desarrolladores Angular. Página oficial angular2. https://v2.angular.io/docs/ts/latest/guide/architecture.html. [Consulta: 9 de Mayo de 2017].

[2] Desarrolladores de PHP. ¿que es php? http://php.net/manual/es/intro-whatis.php. [Consulta: 9 de Mayo de 2017].

[3] Desarrolladores HTML. Documantación de html5. https://developer.mozilla.org/en-US/docs/Web/HTML. [Consulta: 9 de Mayo de 2017].

[4] Desarrolladores Twig. Página oficial twig. https://twig.sensiolabs.org/doc/2.x/. [Consulta: 9 de Mayo de 2017].

[5] Grupo de trabajadores de IBM. Documantación de api rest. https://www.ibm.com/support/knowledgecenter/es/SSMKHH_10.0.0/com.ibm.etools.mft.doc/bi12017_.htm. [Consulta: 23 de Mayo de 2017].

[6] Mario Araque, UX Manager. El proceso, los eventos de scrum. https://www.wearemarketing.com/blog/metodologia-scrum-que-es-y-como-funciona. [Consulta: 15 de Mayo de 2017].